Projekt i programmering 1 ”Bussen”

En stor del av kursen i programmering 1 handlar om att du som elev ska kunna skapa ett större komplett program som en del i ett *projekt*. Vad vi menar med projekt inom ramen för denna kurs är att vi vill simulera hur ett enkelt IT-projekt kan gå till, och där själva kodandet bara är en del av arbetet.

I detta kompendium beskrivs hur projektet ”Bussen”. Notera alltså att man inte *måste* göra detta projekt, men kan upplevas enklare eftersom just detta projekt är tydligt beskrivit nedan, med tydliga krav för respektive betyg.

I projektet bedöms alltså det helhetliga arbetet med planering, genomförande, felsökning och utvärdering.

Notera att för att få betyget *A* (exempelvis) så måste man uppnå kunskapskraven för *alla* moment för detta betyg. Det räcker alltså inte att göra ett bra kodningsarbete.

I detta kompendium beskrivs upplägg, programkrav, förslag på projekt och riktlinjer om du själv väljer ett projekt.

Sedan har vi ett kodskal som underlättar ert arbete mycket och som är utformat för detta projekt.

Följande är den arbetsgång som bör följas:

Projektidé

Om du inte väljer att göra ”Bussen” som är projektförslaget i denna kurs, så måste du ha en idé runt vad ditt projekt ska handla om. Ta del av kunskapskraven och krav på produkten nedan. I produktkraven kan du läsa vilka krav som ställs på koden – detta är vägt mot kunskapskraven hos skolverket och där bland annat kraven på komplexitet är olika beroende på målbetyg.

Planering

När ni har en tydlig bild av vad ni ska skapa för något och vilka datastrukturer & variabler, metoder, algoritmer (beräkningar) och kontrollstrukturer som behövs, så ska programmet planeras.

Er planering ska beskriva ett program som motsvarar ert målbetyg.

Läs mer om planering i ett separat avsnitt nedan.

Genomförande

I denna fas ska ni skapa ert program – koda helt enkelt. Utöver att programmet fungerar som det ska, ställs också krav på *hur* ni kodar. Ta del av kunskapskraven nedan. Tänk exempelvis på kodstruktur, lämpliga identifierare och hur ni presenterar programmet för användaren.

Testning och buggar

När programmet kan betecknas som klart ska ni testa det. Troligtvis kommer ni stöta på en hel del *buggar* – glöm inte att de logiska felen kan vara svåra att upptäcka. Ni ska *debugg:a* koden som man lite slarvigt säger.

Visste ni förresten varför det heter *computer bug?*

Det beror på att en dator av den lite äldre sorten (MARK II och 40-talet) gav fel svar. Teknikerna kunde inte hitta felet förrän de öppnade datorn och hittade en nattfjäril (insekt, engelska ”bug”) som störtden elektromekaniska datorn.

Utvärdering

Det är inte bara i skolans värld som man utvärderar projekt, det gör man även i ”riktiga” IT-projekt. Det är viktigt att man kritiskt granskar sitt eget arbete och förmedlar det skriftligt, och där man kopplar till den ursprungliga planen, hur väl programmet fungerar och vilka fel som uppstod vid testningen och hur dessa löstes. Syftet med detta är givetvis att man ska bli en bättre programmerare och inte göra om samma misstag igen.

Planering

Ni ska skapa någon form av planering till ert projekt.

Då är det lämpligt att skapa följande:

1. Beskrivande del där man redogör för hur man tänkt bygga programmet och exempelvis vilka klasser och metoder som ska finnas med. Man kan också tänka sig att man vill formulera särskilda utmaningar som man ännu inte vet fullt ut hur man ska lösa.

2. Översiktsbild (klassdiagram). Man bör också göra ett diagram som visar de klasser med variabler med tillhörande variabler och metoder som ska användas. Ungefär så som det illustreras i denna projektbeskrivning nedan. Detta sätt att planera är egentligen mest lämpligt om man skapar fler än en klass (se krav på betyg längre ner). Figuren ska alltså visa vilka klasser som ska användas och hur de ska konstrueras – en *modell* över programmet.

3. Någon pseudo-kod. Man kan exempelvis välja ut en eller ett par metoder att beskriva med pseudo-kod. Undvik att beskriva en metod som enbart visar en meny exempelvis.

4. Något aktivitetsdiagram (sekvensdiagram). Man kan exempelvis välja ut en eller ett par metoder att beskriva med aktivitetsdiagram. Undvik att illustrera en metod som enbart visar en meny exempelvis.

Denna uppgift är alltså relativt öppen i sitt upplägg men desto tydligare och bättre du kan förmedla hur programmet ska konstrueras, desto bättre betyg och desto lättare för handledaren att ge vettig feedback.

Bussen

Detta är ett förslag på projekt och där det konkret beskrivs vad programmet ska innehåll och fungera utifrån ditt målbetyg.

Bakgrund

Du ska göra ett skolprojekt där du ska registrera personer och dess ålder samt kön, som sätter sig och går av en buss. För detta ändamål tänker du att du ska använda dina kunskaper i programmering.

Utifrån denna buss ska du kunna räkna fram exempelvis genomsnittlig ålder, samt hur många manliga och kvinnliga passagerare som finns.

Om du är ännu mer äventyrlig (högre betyg) så kommer också fler saker kunna hanteras (se nedan).

Vi kan i detta fall göra det enkelt för oss och att denna undersökning sker i ett litet samhälle där ”alla känner alla”.

Klassen Buss och objektet

Notera att det finns ett skal till uppgiften som ni kan utgå ifrån. Detta ska ni få tillgång till via er läroplattform. I detta kod-skal finns rikligt med kommentarer.

För att det inte ska bli förvirrande som det lätt kan bli när man jobbar i program-klassen (den som innehåller den statiska metoden *main* som alltid startar först) så ska vi skapa en klass som kontrollerar programmet. Denna klass kan också innehålla eventuella publika och/eller privata variabler som underlättar körning av programmet.

Vi skapar ett objekt av klassen *buss* med namnet minbuss (eller något annat namn – den kan heta vad som helst som ”tallkotte” eller ”kokosboll” även om det inte är så lämpligt.)

Det innebär att koden i programklassen ser ut så här (ur kodskalet):

```
class Program
{
    public static void Main(string[] args)
    {
        //Skapar ett objekt av klassen Sodacrate som heter sodacrate
        //Denna del av koden kan upplevas väldigt förvirrande. Men i sådana fall är det bara att "skriva av".
        var minbuss = new Buss();
        minbuss.Run();
        Console.Write("Press any key to continue . . . ");
        Console.ReadKey(true);
    }
}
}
```

“minbuss” här är alltså en identifierare för ett objekt som skapats utifrån klassen Sodacrate.

I klassen Buss finns då en metod som heter Run () som innehåller koden som körs när programmet startar, inklusive huvudmenyn.

```
public void Run()
{
    Console.WriteLine("Welcome to the awesome Buss-simulator");
    //Här ska menyn ligga för att göra saker
    //Jag rekommenderar switch och case här
    //I filmen nummer 1 för slutprojektet så skapar jag en meny på detta sätt.
}
```

Krav på Bussen

Detta är krav på konkret innehåll i programmet. För krav gällande exempelvis kodstruktur, kommentering, och formatering av utdata hänvisas till de formella kunskapskraven sist i detta kompendium.

Kraven för respektive betyg beskrivs som kommentarer i kodskalet.

Betyg E

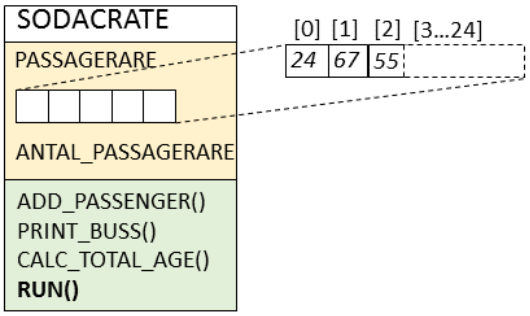
I objektet *minbuss* ska det finns en vektor som håller reda på 25 stycken passagerare. Varje element i vektorn är en ålder.

Observera att det ska vara just en **vektor** och *inte* en lista.

I *sodacrate* ska det finnas en switch-case meny där alternativen kopplas till olika metoder. Dessa metoder är:

1. Lägga till en passagerare
2. Skriv ut bussen, eller rättare sagt – skriv ut alla åldrar på passagerare.
3. Beräkna den totala åldern av alla passagerare.
4. Avsluta programmet

Under metoden *Run()* ska alltså kod skrivas som skriver ut text som välkomnar användaren och därefter får en lista över vilka saker man kan göra. Detta val bör ju läggas i en loop så inte programmet stänger ner efter att man valt något.



Man kan själv välja hur stor vektorn ska vara.

Man ska inte kunna stoppa in fler passagerare än vad som får plats.

Är bussen full ska man få meddelande om detta och passageraren kan inte stiga på.

Betyg C

För betyget C ska klassen *Sodacrate* byggas ut med ytterligare metoder såsom ni kan se i kodskalet.

Betyg A

Istället för att vektorn lagrar *tal* så ska nu vektorn lagra objekt som representerar passagerare. Du ska alltså skapa en ny klass som förslagsvis heter *passenger* eller något liknande

Den nya klassen kan nu hantera fler värden för en kund och följande värden ska finnas:

1. Ålder
2. Kön
3. Annan eventuell valfri data

Den nya klassen ska också ha en metod som bestämmer hur passageraren beter sig när man petar på honom eller henne. Hur passageraren beter sig kan då baseras på ålder och kön.

Mer objektorientering (svår)

För en guldstjärna i uppgiften ska du sätta variablerna i klassen *soda* och *sodacrate* som privata och skapa en korrekt konstruktor (för klassen soda) samt andra nödvändiga metoder som för att exempelvis returnera pris. Använd gärna annat som jag visar i filmen också, såsom ToString()-metoden.

Hjälpkod och övriga instruktioner

Ps Det finns några begrepp som istället knyter an till en läskback. Principen är precis samma.

Detta avsnitt är till för att ge viss grundläggande startkod

//Skapa en vektor med plats för 25 stycken objekt av klassen *passenger:*

```
passenger[] passagerare = new passenger[25];
```

//Skapa ett objekt av klassen *dryck* i vektorn i position 4:

```
passagerare[4] = new dryck();
```

//Skapa ett objekt av klassen *dryck* i vektorn i position 4 som heter ”Cola”, kostar 5 kr och är av typen ”soda”. Detta förutsätter att klassen dryck har en *konstruktor* med inparametrar som matchar. Se klassen dryck nedan:

```
passagerare[4] = new passenger(56, ”man”);
```

//Byta position på två drycker i backen.

```
dryck tmp = new dryck();
tmp = min_back[4];
min_back[4] = min_back[5];
min_back[5] = tmp;
```

```
//Skriva ut alla drycker i backen. Är platsen tom skrivs det ut. Här används null som betyder noll, inget, tomt eller vad man nu vill..
```

```
foreach(var dryck in min_back)
 {
     if (dryck != null)
     Console.WriteLine(dryck);
     else
     Console.WriteLine("Tom plats");
 }
```