Projekt i programmering 1 (ver 2)

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 ”Läskbacken”. 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.

Följande är den arbetsgång som bör följas:

Projektidé

Om du inte väljer att göra ”läskbacken” 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 stört den 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. 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 pseudokod.

4. Något aktivitetsdiagram (sekvensdiagram). Man kan exempelvis välja ut en eller ett par metoder att beskriva med aktivitetsdiagram.

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.

Läskbacken (projektförslag)

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

Bakgrund

Varje månad lunkar du ner till närköpet bakom hörnet och köper en dryckesback som du fyller med olika drycker – vatten, lättöl och läsk. Eftersom du är halvt ohälsosamt intresserad av läskbackar och vad du stoppar i läskbacken har du bestämt dig för att skriva ett program som kan administrera din läskback. Spontant tänker du att programmet ska kunna lagra tjugofyra stycken drycker i en vektor och därefter ska du skapa några metoder som att söka efter en dryck baserat på varumärkesnamn, räkna ut totalpriset för backen samt stoppa i och ta bort drycker ur backen.

Du har sedan tankar på hur du kan göra programmet *ännu* bättre men det kommer ta längre tid och kräva mer studier tänker du, så programmet ska vara omfattande i en lagom mängd:

Klassen Sodacrate och objektet

Notera att det finns ett skal till uppgiften som ni kan utgå ifrån. Detta ska ni få tillgång till via er läroplattform men kan också hittas via denna länk. I denna fil 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 *sodacrate* med namnet sodacrate (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:

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

```
13      public class Program
14          {
15          public static void Main(string[] args)
16              {
17              Sodacrate sodacrate = new Sodacrate();
18              sodacrate.Run();
19              }
20          }
21
```

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

```
namespace test
{
    public class Sodacrate
    {
    //Eventuella fält ( variabler)

    public void Run ()
    {
    Console.WriteLine("Welcome to the awesome sodacrate-simulator");
    //Här startar programmet
    }
}
```

Krav på läskbacken

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.

I uppdragsbeskrivningen på Hermods finns en hjälpkod man kan ladda ner och använda som mall för att komma igång.

Betyg E

I objektet *sodacrate* ska det finns en vektor som håller reda på 25 stycken flaskor. Varje element i vektorn är ett namn som exempelvis ”coca-cola” eller ”Ramlösa”.

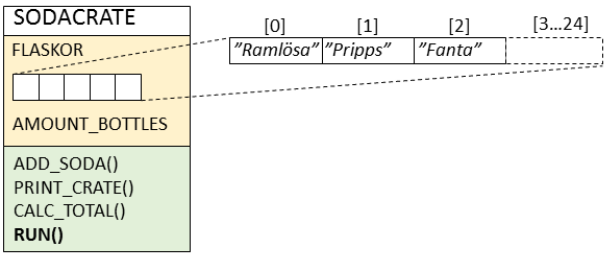
Observera att det ska vara just en **vektor** och *inte* en lista.

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

1. Lägga till en läsk i vektorn (läskbacken)
2. Skriva ut innehållet i vektorn (läskbacken)
3. Beräkna det totala värdet av backen och skriv ut på skärmen
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 ska inte kunna stoppa in fler än 25 drycker i backen, eventuellt kan man få välja att ersätta en flaska i en full back med en annan. Här används då med fördel variabeln *amount_bottles* (se figuren) som kan hålla reda på antal flaskor.



Är backen full ska man få meddelande om detta och då får man välja en position och den gamla läsken försvinner.

För uträkning av det totala värdet så utgår vi ifrån att varje flaska kostar 5 kronor.

Betyg C

För betyget C ska klassen *Sodacrate* byggas ut med ytterligare metoder:

1. En metod som anropas för att söka efter en flaska baserat på namn. Detta beskrivs i läroboken och kapitel 13 (kodexempel för sökning på sidan 149). I denna metod kan man också ha en *inparameter* som då är den sträng som du ska söka efter.

2. En metod som ska anropas för att *sortera* vektorn med läskflaskor utifrån namnet. Eftersom alla flaskor har samma pris (för betyget C) får vi istället sortera efter hur många tecken det finns i namnet – dvs hur långt namnet är. Själva sorteringen beskrivs i läroboken och kapitel 13 (kodexempel för sortering på sidan 159). Gör man uppgiften enligt A kan man sortera efter pris också vilket är roligare kanske.

Tänk följande kod (från Microsfts sida):

```
string str = "abcdefg";
Console.WriteLine("1) The length of '{0}' is {1}", str, str.Length);
```

Här kommer 7 (sju) att skrivas ut i konsollen då strängen ”str” har sju tecken i sig. På detta sätt kan man sedan jämföra två strängars längd med varandra och vilken sträng som är flest tecken för bubbelsorteringen.

Notera då att huvudmenyn behövs byggas ut med dessa funktioner (sortering och sökning).

Struct?

I hjälpkoden finns ett förslag att man kan skapa en *struktur.* Det är en ”klass light” där man kan ”bunta ihop” flera värden under ett namn. Då kan man sortera efter pris exempelvis och uppgiften blir roligare att jobba med. Mer information i hjälpfilen.

Betyg A

Istället för att vektorn lagrar *namn* så ska nu vektorn lagra objekt som representerar flaskorna. Du ska alltså skapa en ny klass som förslagsvis heter *bottle, flaska* eller något liknande.

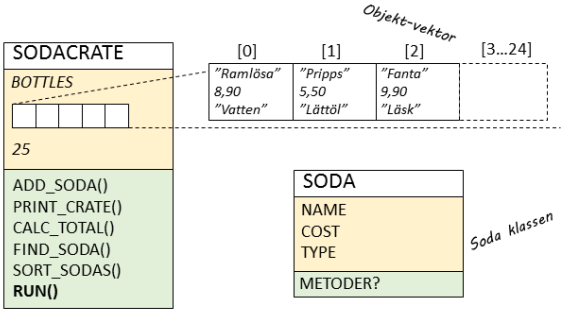
Den nya klassen kan nu hantera fler värden för en flaska och följande värden ska finnas:

1. Namn
2. Kostnad (kostnad för flaskan i kronor)
3. Typ (Läsk, vatten eller lättöl)

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

4. Namn
5. Kostnad (kostnad för flaskan i kronor)
6. Typ (Läsk, vatten eller lättöl)

Notera nu också att förutsättningarna för beräkning av genomsnittligt pris förändras eftersom flaskorna kan ha olika pris.



Konstruktor (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.

Hjälpkod och övriga instruktioner

Detta avsnitt är till för att ge viss grundläggande startkod

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

```
dryck[] min_back = new dryck[25];
```

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

```
min_back[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:

```
min_back[4] = new dryck(”Cola”, 5, ”soda”);
```

//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");
 }
```

Andra förslag på projekt

Nedan kommer några andra förslag på projekt.

Det är viktigt att du förankrar med den rättande läraren vad du tänkt jobba med så att ditt projekt är förenligt med dina betygsmål.

Joppes hunddagis

Istället för en läskback så jobbar du med ett ”hunddagis” och där du har en klass som motsvarar en ”hund”. Hunddagiset ska ha en vektor som kan lagra ett fast antal hundar – varje position blir då hundens egna ”bås”.

Det roliga med hundar är att de är levande än läskflaskor och man kan ha en metod exempelvis som styr hur hunden beter sig när en person knackar på dörren.

I övrigt kan man jobba på ett liknande sätt och där man exempelvis kan beräkna den genomsnittliga åldern på hundarna (med decimaltal).

Bussen

Detta projekt går ut på att man istället övervakar en buss och beroende på betygsmål kan man antingen representera passagerare med ålder (betyg E) medan betyget A innebär att varje person är ett eget objekt. Här öppnas spännande möjligheter då man kan lagra information om kön, fysiska egenskaper och ålder exempelvis, och använda det för någon form av redovisning av statistik. För det högsta betyget är det lämpligt att man kan lagra flera bussar (men förmodligen inget måste)

Bingo

Denna uppgift går ut på att göra ett Bingo-spel. Denna uppgift ska kretsa runt att en spelare har en bingo-bricka. I början av programmet ska man få välja hur många spelare som ska vara med och datorstyrda motståndare ska få slumpmässiga tal valda. Den som får flest poäng vinner. Denna uppgift kan bli svår att göra om man siktar mot betyget *E* men för ett högre betyg kan det bli bra – särskilt om man kan spela flera omgångar i samma programkörning.

Varje bingo-bricka ska vara ett eget objekt (från en klass *bingo-bricka*) och om man siktar mot ett högre betyg. Klassen ska beskriva att varje objekt ska ha en vektor för talen, tidsstämpel samt eventuella metoder. En metod kan vara att talen som ska markeras på bingo-brickan ska slumpas.

Mer förslag

Har ni som elever kommit på en bra projektidé så kan dessa komma att fyllas in här.

Programkrav

Detta specifika avsnitt är främst tänkt för handledare och lärare.

Nedan följer vilka krav på vad som kodmässigt ska ingå för de olika betygsmålen. Detta är viktat mot de formella kunskapskraven i kursen och exempelvis krav på komplexitet. I projektet ”Läskbacken” som beskrivs så konkretiseras dessa krav i kommande avsnitt. Viss kodhjälp ska tillhandahållas för en grundläggande funktionalitet för klasserna.

Notera att den nedan endast beskrivs rent kvantitativt *vad* som ska ingå – i de formella kunskapskraven (som finns i slutet på detta kompendium) anges exempelvis krav på kvalitet i kodandet.

Betyg E

Följande ska ingå i programmet.

- ✓ Skapa och använda objekt baserat på någon klass
- ✓ Några kontrollstrukturer
- ✓ Några metoder däribland med funktionalitet för
    - Enkel sökning i en vektor
    - Enkel hantering av vektorn
- ✓ Datastrukturer som objekt, vektor och variabler

Betyg C

Följande ska ingå i programmet

- ✓ Skapa och använda objekt baserat på någon klass
- ✓ Flera kontrollstrukturer
- ✓ Flera metoder däribland med funktionalitet för
    - Söka i en vektor
    - Hantera en vektor
    - Sortera en vektor
- ✓ Datastrukturer som objekt, vektor och variabler

Betyg A

Följande ska ingå i programmet

- ✓ Komplex karaktär
- ✓ Skapa och använda några klasser med mer komplexa samband
- ✓ Flera kontrollstrukturer
- ✓ Grundläggande objektorienterad programmering
- ✓ Datastrukturer som objekt, listor och variabler
- ✓ Flera metoder däribland med funktionalitet för
    - Söka i en lista och en vektor
    - Sortera och hantera en vektor och/eller en lista

Formella kunskapskrav

Här kommer de formella kunskapskrav från skolverket som gäller för detta projekt

Betyg E

Eleven formulerar och planerar programmeringsuppgifter med pseudokod **utifrån en förlaga eller, i samråd med handledare**, med aktivitetsdiagram. I planeringen väljer eleven **med viss säkerhet** kontrollstrukturer, metoder, variabler, datastrukturer och algoritmer som är adekvata för uppgiften.

Eleven implementerar **en** sökningsalgoritm och **i samråd** med handledare också **en** sorteringsalgoritm **eller en** rekursiv algoritm. I sin programmering skriver eleven en korrekt, **delvis** strukturerad och kommenterad källkod, med konsekvent kodningsstil och tydlig namngivning.

Dessutom väljer eleven **med viss säkerhet** ett uttryckssätt som är anpassat för att på ett **tillfredsställande**sätt interagera med den avsedda användaren.

Elevens färdiga program eller skript är utfört med **tillfredsställande** resultat i ett eller flera programmeringsspråk och innehåller sekventiell programmering och grundläggande objektorienterad programmering som är stabil och robust **i program av enkel karaktär**.

Eleven anpassar **med viss säkerhet** sin planering av programmeringsuppgiften och utför felsökning av **enkla** syntaxfel. Innan programmeringsuppgiften avslutas utvärderar eleven med **enkla** omdömen programmets prestanda och ändamålsenlighet i **någon** situation **eller** i **något** sammanhang.

Eleven kommunicerar **med viss säkerhet** med datalogiska begrepp om programmeringsuppgiften och dess utvärdering.

När eleven samråder med handledare bedömer hon eller han **med viss säkerhet** den egna förmågan och situationens krav.

Betyg C

Eleven formulerar och planerar programmeringsuppgifter med pseudokod eller med aktivitetsdiagram. I planeringen väljer eleven **med viss säkerhet** kontrollstrukturer, metoder, variabler, datastrukturer och algoritmer som är adekvata för uppgiften.

Eleven implementerar **en** sökningsalgoritm och **efter samråd** med handledare också **en** sorteringsalgoritm **och en** rekursiv algoritm.

I sin programmering skriver eleven en korrekt, strukturerad och kommenterad källkod, med konsekvent kodningsstil och tydlig namngivning. Dessutom väljer eleven **med viss säkerhet** ett uttryckssätt som är anpassat för att på ett **tillfredsställande** sätt interagera med den avsedda användaren.

Elevens färdiga program eller skript är utfört med **tillfredsställande** resultat i ett eller flera programmeringsspråk och innehåller sekventiell programmering och grundläggande objektorienterad programmering som är stabil och robust.

Eleven anpassar **med viss säkerhet** sin planering av programmeringsuppgiften och utför **på ett systematiskt sätt** felsökning av syntaxfel, **körtidsfel och programmeringslogiska fel**.

Innan programmeringsuppgiften avslutas utvärderar eleven med **nyanserade** omdömen programmets prestanda och ändamålsenlighet i **några** situationer **och** sammanhang.

Eleven kommunicerar **med viss säkerhet** med datalogiska begrepp om programmeringsuppgiften och dess utvärdering.

När eleven samråder med handledare bedömer hon eller han **med viss säkerhet** den egna förmågan och situationens krav.

Betyget A

Eleven formulerar och planerar programmeringsuppgifter med pseudokod eller med aktivitetsdiagram. I planeringen väljer eleven **med säkerhet** kontrollstrukturer, metoder, variabler, datastrukturer och algoritmer som är adekvata för uppgiften **samt motiverar utförligt sina val**.

Eleven implementerar sökningsalgoritmer och **efter samråd** med handledare också sorteringsalgoritmer **och** rekursiva algoritmer. I sin programmering skriver eleven en korrekt, strukturerad och **utförligt** kommenterad källkod, med konsekvent kodningsstil och tydlig namngivning. Dessutom väljer eleven **med säkerhet** ett uttryckssätt som är anpassat för att på ett **gott** sätt interagera med den avsedda användaren.

Elevens färdiga program eller skript är utfört med **gott** resultat i ett eller flera programmeringsspråk och innehåller sekventiell programmering och grundläggande objektorienterad programmering som är stabil och robust **i program av komplex karaktär**
.
Eleven anpassar **med säkerhet** sin planering av programmeringsuppgiften och utför **på ett systematiskt och effektivt sätt** felsökning av syntaxfel, **körtidsfel och programmeringslogiska fel**.

Innan programmeringsuppgiften avslutas utvärderar eleven med **nyanserade** omdömen **och med förslag på förbättringar** programmets prestanda och ändamålsenlighet
i **flera** situationer **och** sammanhang.

Eleven kommunicerar **med säkerhet** med datalogiska begrepp om programmeringsuppgiften och dess utvärdering.

När eleven samråder med handledare bedömer hon eller han **med säkerhet** den egna förmågan och situationens krav.