Programmeringsprojekt, STS1

1. Inledning

Nedan följer ett antal större programmeringsuppgifter.

En av dessa uppgifter ska lösas och senare presenteras på ett seminarium (se särskilda anvisningar för detta). Egna idéer till uppgifter är välkomna.

Uppgiften ska lösas i grupper om max två personer. Det maximala antalet grupper som får välja en och samma uppgift är begränsat.

2. Redovisning

En skriftlig rapport med programlistningar ska lämnas in i *tre* exemplar senast 23 maj 10.15 till Anders Berglund (Obs!), postfacket på plan 4, hus 1, vid kopiatorerna. Anders tilldelar varje grupp en seminarietid. Önskemål om en viss tid accepteras inte, annat än av mycket starka skäl, såsom exempelvis att någon vill undvika en viss tid pga. av ett sedan länge inplanerat läkarbesök. Den som lämnar sådana önskemål ska på begäran kunna visa dokument som styrker argumentet.

Muntligt samarbete mellan grupper är tillåtet och rekommenderat, så länge olika uppgiftslösningar "inte liknar varandra". Samarbete med grupper som löser samma uppgift ska redovisas i rapporten.

I rapporten ska åtminstone följande punkter ingå:

- En beskrivning av uppgiften
- En översikt av uppgiftslösningen
- Begränsningar i den genomförda lösningen
- Antaganden utöver uppgiftsformuleringen samt förklaringar till varför dessa antaganden gjorts och varför de är rimliga
- Kortfattad redogörelse för muntligt samarbete med grupper med samma uppgift
- Beskrivning av och motivering till valda klasser
- Beskrivning av metoder
- Förslag till förbättringar, vidareutveckling etc.
- Erfarenheter och slutsatser
- Användarhandledning
- Anvisningar till någon som vill provköra ert program
- Uppgifter om var filerna finns och vad den heter. Tänk på accessrättigheterna. Dina filer ska vara läsbara för samtliga, *när projektet är klart*. Det innebär att samtliga dina

kurskamrater ska kunna läsa och ladda hem dessa. Kontrollera gärna att du har gjort rätt, genom att be en kurskamrat kontrollera, att du har satt läsrättigheterna rätt. Felaktiga läsrättigheter, så att dina kurskamrater inte kan läsa din *färdiga uppgift vid inlämningen* innebär automatiskt att uppgiften underkänns

Dessutom ska programlistningar lämnas in.

Rapporten får skrivas på svenska eller engelska.

3. Uppgiftslösningen

Lösningen ska spegla god programmeringssed för objektorienterad programmering, och ska på ett relevant sätt spegla klassbegreppet.

Du behöver inte använda en mer avancerad in och utmatning än den som ges av paketet `extra.*` annat än för de uppgifter där det *uttryckligen anges* att ett grafiskt eller mer avancerat användarsnitt ska användas. Du behöver heller inte kontrollera indata annat än i de uppgifter då detta anges som ett krav.

Du kan också göra din bildskärmshantering mycket enkel. Det är tillfyllest om skärmen skrivs om för varje drag i samband med spel, reservation i samband med bokning etc. som visas, om inte uppgiftstexten specificerar något annat.

Många uppgifter blir lättare att lösa eller, i vissa fall, kräver programspråkskonstruktioner som inte är centrala i kursen och som inte är detaljstuderade i undervisningen. Exempel på sådana konstruktioner är arrayer med fler dimensioner än en (såsom matriser) eller arrayer som innehåller objekt ur klasser. Inga av dessa konstruktioner är komplicerade, men en lässtund i Skansholm kan rekommenderas. Olika "exotiska" eller "märkliga" konstruktioner, som förvisso existerar i Java behövs inte för någon uppgift och rekommenderas inte heller.

Bestäm dig för en ambitionsnivå och håll dig till den. Att låta uppgiften växa volymmässigt ger dig inte några större meriter, att i stället göra ett dåligt arbete innebär att uppgiften måste kompletteras eller blir underkänd och skapar extra, onödiga problem.

4. Ett gott råd

Vi rekommenderar varmt att du diskuterar din lösningsstruktur (val av klasser, viktiga metoder och fält) med Elena, innan du börjar koda större avsnitt. På så sätt kan du säkert undvika många strukturella problem med din lösning.

5. Uppgifter

Uppgifterna nedan är skissartat formulerade. Du kommer att själv vara tvungen att göra antaganden av olika slag. Dessa ska redovisas. Rimliga antaganden, som du kan förklara och motivera vid en fråga godkänns.

5.1 Egen uppgift

Välj själv en uppgift. Diskutera gärna med Anders. Det går bra att ringa på 070 425 02 11 torsdag kväll.

5.2 Fia med knuff

Implementera det gamla brädspelet "Fia med knuff" med en datormotståndare.

I spelplanen nedan har blå inte fått ut någon pjäs, medan Gul har fått ut tre varav en redan är i boet.

```
          000
          0r0
  G = 3   0rY    R = 3
          0r0
          Rr0
0 0 0 0 0 0 0 0 0 Y 0 0
0 g g g g     y Y y y 0
0 0 G 0 0 0 0 0 0 0 0 0
          0b0
          0b0
  B = 4   0b0      Y = 1
          0b0
          000
```

Reglerna är enkla: Plupparna för sig moturs, kan knuffa ut motståndarens pluppar, man får endast gå ut på en 6a. Man får slå ett extra slag på en 6a (dock högts tre gånger i rad). Den spelare som först får in sina pluppar i boet har vunnit.

Datormotståndaren behöver inte vara smart, utan ska enbart hålla sig till reglerna.

Människospelare (tre stycken) ska dock kunna välja vilken plupp de flyttar på, ta eller gå in i boet med.

Programmet ser ut vinnare samt håller reda på hur många pluppar som är inne

För den ambitiöse:

- Skriv en snyggare bildhantering
- Gör en smart datormotståndare
- Låt datorn kunna spela för en två eller tre personer, beroende på hur många människor som vill spela.
- Programmet ser till att reglerna följs.

5.3 Platsbokning på SJ

Skriv ett program som hjälper SJ med platsbokning i en järnvägsvagn. När programmet startas ska användaren ange vilken vagn som ska bokas. (Systemet kan således enbart göra bokningar i en vagn i taget.) På skärmen ska bilden nedan (eller en liknande bild) ritas upp. Bokade platser ska sedan markeras på ett rimligt sätt.

```
---------------------------------------
| 1   5   9  13   |   17  21  25  29 |
| 2   6  10  14   |   18  22  26  30 |
|    Tyst kupé    | Nalle tillåten   |
| 3   7  11  17   |   19  23  26  31 |
| 4  10  12  18   |   20  24  28  32 |
---------------------------------------
```

Man bokar genom att fylla i ett eller flera platsnummer, eller genom att ange önskemål (såsom fönster mm) om platserna. Därefter ritas skärmen om med det aktuella bokningsläget. När man inte vill boka mer, skrivs en eller vid behov flera platsbiljetter ut (på bildskärmen) med ungefär följande utseende:

```
Platsbiljett
Sth - Göteborg 9.05
Plats 19
Nallekupé
Gång
```

5.4 Räkna med stora tal

Ta fram ett program som kan räkna med stora tal. Begränsningar, om de införs, ska ha sin utgångspunkt i radlängder vid inmatning eller dylikt, inte i lagringskapacitet i en viss variabler av en viss typ.

Programmet ska klara addition, subtraktion och multiplikation.

Exempel på dialog (användarens svar är understrukna):

```
Räknesätt: +
Tal 1: 87345171325167
Tal 2: 5319213411
87345171325167 +  5319213411 =  873457032465078
```

Ledning: Klassen `Vector` är värd att studera ☺

5.5 Spelet Life

En familj av encelliga organismer lever i en platt (tvådimensionell) värld. Celler föds och dör enligt givna regler. Beroende på utgångsläget kan en koloni expandera, dö ut eller nå ett stabilt tillstånd.

Uppgiften består i att göra en simulator där man simulerar en godtycklig uppsättning celler i världen. I en tvådimensionell värld har celler (normalt) åtta platser för grannar. Du bör kunna hantera minst 20* 20 = 400 celler. Simuleringen bör skrivas ut på skärmen (på ett enkelt sätt) efter varje steg.

Levnadsregler för cellerna:

- En cell överlever till nästa generation om den har två eller tre grannar. Har den färre dör den av ensamhet, om den har fler dör den av överbefolkning och svält.
- I en tom cell, som har exakt tre grannar kommer en ny cell att födas till nästa generation.
- Cellerna föds och lever bara i den simulerade världen, inte utanför den.

Några kul starttillstånd för en koloni:

```
 *  *
 ****
*    *                                        *
* ** *                  *                       *
*    *                    *                  *    *
 ****                   ***                 *****
```

Cheshirekatten från Alice i underlandet. Efter ett par generationer återstår den grinande munnen	Glidare Rör sig snett uppåt	Rymdskeppet Rör sig i sidled.

5.6 Romerska årtal

Skriv ett program som omvandlar "siffertal" till romerska årtal och tvärtom. Årtalen ska kunna variera från 1 till 3999. Systemet ska reager ordnat på inmatning av större årtal eller år 0.

```
Vill du konvertera till romerskt eller till arabiskt [r/a]? r
Årtal:      1990
1990 skriv MCMXC
Vill du konvertera ett nytt tal [j/n]? n
```

De romerska talen skrivs I, V, X, L, C, D, M och motsvarar 1, 5, 10, 50, 100, 500, 1000. Kombinationer som IV betecknar 4, IX betyder 9, XC 90 etc. I övrigt adderas bokstäverna. Läs gärna mer, exempelvis i Nationalencyklopedin.

För den ambitiöse:
Låt programmet avgöra om det inmatade talet är romerskt eller ett tal (arabiskt). Denna utveckling är relativt enkel. Ledning: Titta på klassen `String` och/eller Skansholms paket `extra`.

5.7 Spelet talgiss

Spelet talgiss (dvs. tal-giss, gissa ett tal) går till på följande sätt:

Programmet slumpar fram ett positivt fyrsiffrigt tal, där alla siffror är olika och skilda från noll. Det är detta tal som "datorn tänker på". Nu gäller det för användaren att gissa sig fram till detta tal, med utgångspunkt från den information och hjälp som datorn ger efter varje gissning.

Datorns hjälp markeras på följande sätt:

R	innebär att en gissad siffra är rätt och rätt placerad
F	innebär att en gissad siffra finns, men står på fel plats
.	(punkt) innebär att en gissad siffra inte finns med i talet.

Informationen skriv ut i ordningen:

1. Alla eventuella R
2. Alla eventuella F
3. Alla eventuella .

Man får alltså inte veta vilken siffra som är rätt eller fel.

Exempel på körning (användarens inmatning är understruken):

```
Datorn har gissat.
Din gissning: 1234
1234: RF..
Din gissning: 1278
1278: RR..
Din gissning: 3285
3285: RRFF
Din gissning: 3258
3258: RRRR
Rätt efter fyra gissningar.
Vill du spela igen? [j/n]: n
```