



Inlämningsuppgift 6 – Rörlig text och arrayer

1 Inledning

Inlämningsuppgift 6 ska lösas i grupp med 5-7 medlemmar. Uppgiften ska bidra till en grundläggande förståelse för:

- Design och implementation av ett java-program
- Hantering av arrayer i en och två dimensioner
- Skapa en enkel app i Android-miljö (ej obligatoriskt)
- Planera och utföra testning av kod
- Jobba i grupp

1.1 Utrustning

Verktyg som behövs för denna programmeringsuppgift:

- Eclipse
- Android Studio (Utvecklingsverktyg för Android, på PC/Mac/Linux)
- Android-telefon (optionell)

1.2 Klasser som bifogas

Klasserna `ColorDisplay`, `IntDisplay` och `Color` bifogas för utveckling av systemet på dator.

Klasserna `ColorDisplay` och `IntDisplay` bifogas för utveckling av systemet på android-enhet.

I bilaga 1 beskrivs klasserna.

2 Beskrivning av uppgiften

I den här uppgiften skall ni skapa ett antal klasser för att hantera arrayer i en och två dimensioner, testa dessa och sedan sätta ihop dem till ett system, som kan köras på PC och i en telefon med Android.

Basen i uppgiften är ovanstående klasser tillsammans med en klass som hanterar färg förpackade som *int*-värde. Den två-dimensionella arrayen ska ha 7x7 element och den en-dimensionella arrayen ska ha 7 element. Dessutom kommer det ingå UI-klasser mm i systemen.

Målet med uppgiften är att kunna läsa en text från ett inmatningsfönster och låta texten från denna "rinna" från höger till vänster (precis som i reklamskyltar på stan och information i bussar och tåg) med hjälp av en grafisk komponent.

Sedan finns ett antal idéer på optionella uppgifter (se Moment 9), vilka ni utför i mån av tid.

För att klara av uppgiften måste ni planera nedanstående moment så att alla i gruppen hjälper till. Tips: Det är bra att använda "par-programmering". Har ni inte redan provat detta, så kan det vara en god ide att göra det nu.

Uppgiften består av följande moment:

1. Skapa klasser för en-dimensionell array och två-dimensionell array.
2. Skapa en testmiljö för `Array7` och `Array7x7` och testa dem utförligt
3. Utöka funktionaliteten i `Array7x7` och testa dem utförligt



4. Använda Array7 och Array7x7 vid hantering av färgelement
5. Designa och implementera alla bokstäver (A-Z, 0-9, etc., så de kan visas i en 7x7 färg-display)
6. Implementera rinnande text från och testa detta
7. Skapa ett klassdiagram över lösningen i moment 6
8. Flytta över till Android och testa
9. Optionella uppgifter

2.1 Syfte och mål

Uppgiften går ut på att bygga och sätta ihop komponenter till ett komplett system. Detta system skall även testas. Då uppgiften utförs i grupp skall även grupparbete tränas.

2.2 Tidsåtgång

Vi räknar med att det finns ca 40 timmar tillgänglig tid för varje student för lösning av uppgiften.

2.3Handledning

- Onsdagen den 19/12, 8 – 12 i G8:423
- Tisdagen den 8/1, 8 – 12 i G8:423

2.4 Examination

2.4.1 Praktisk och muntlig redovisning

Era lösningar (testmiljöer och rinnande text) ska presenteras. Varje gruppmedlem skall beskriva sitt bidrag och samtliga i gruppen skall kunna redogöra för funktionalitet i systemen.

Muntlig presentation äger rum under vecka 3.

2.4.2 Inlämning på Canvas

På Canvas ska gruppen lämna in (senast i samband med muntlig presentation):

- All källkod för lösning på PC i filen java.zip
- All källkod+xml-filer för lösning på android-enhet i filen android.zip (ej obligatoriskt)
- Klassdiagram över lösningen i moment 6.



3 Moment 1: Skapa klasser för en och tvådimensionella arrayer

3.1 Beskrivning

I detta moment skall ni skapa klassen `Array7`, som innehåller 7 element av typen `int`, och klassen `Array7x7`, som innehåller 7x7, dvs 49 element av typen `int`.

3.2 Uppgifter

Uppgift 3.2.1:

Skapa klassen `Array7` som ska innehålla 7 element av typen `int`. Elementen ska lagras i en array av typen `int[]`. Skapa lämpliga metoder för att skriva och läsa enskilda element:

```
setElement(int pos, int value)
getElement(int pos) : int
```

Skapa också lämpliga metoder för att skriva och läsa samtliga element:

```
setArray(Array7 array7)
setArray(int[] array)
toArray() : Array7           // ny instans av Array7 ska returneras
toIntArray() : int[]         // ny instans av int[] ska returneras
```

Skapa slutligen lämpliga konstruktörer, t.ex.

```
Array7()
Array7( Array7 array7 )
Array7( int[] array )
```

Uppgift 3.2.2:

Skapa klassen `Array7x7` som ska innehålla 7x7 element av typen `int`. Skapa lämpliga metoder för att skriva och läsa enskilda element:

```
setElement(int row, int col, int value)
getElement(int row, int col) : int
```

Skapa metoder som kan skriva och läsa rader och kolumner:

```
setRow(int row, Array7 theRow)
getRow(int row) : Array7
setCol(int col, Array7 theCol)
getCol(int col) : Array7
```

Skapa metoder som kan skriva och läsa samtliga element:

```
setArray(Array7x7 array7x7)
setArray(int[][] array)
toArray() : Array7x7
toIntArray() : int[][]
```

Skapa lämpliga konstruktörer. Lägg även till andra metoder som gruppen finner lämpliga.



3.3 Administration

Gruppen bör bestämma hur kod kan delas mellan medlemmarna i gruppen (mail eller gemensam lagringsplats?).

En planering bör tas fram som visar vem som förväntas göra vad, hur mycket tid det förväntas ta och när resultat levereras till övriga gruppmedlemmar.

4 Moment 2: Skapa en testmiljö för klasserna i Moment 1 och testa dessa utförligt

4.1 Beskrivning

I detta moment skall ni skapa en miljö där klasserna i momentet ovan kan testas.

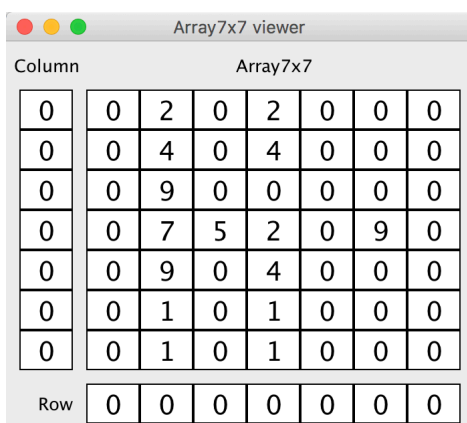
4.2 Uppgifter

Uppgift 4.2.1:

Skapa fönster liknande de nedan. *array7x7* nedan betecknar en *Array7x7*-instans.

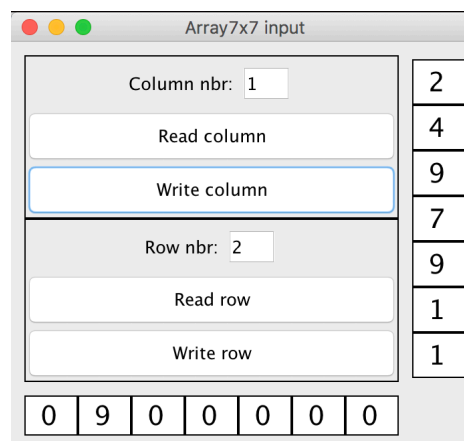
Fönstret **Array7x7 viewer** är till för att visa innehållet i *array7x7* och innehållet i en speciell rad eller kolumn i *array7x7*. JPanel-komponenten i fönstret består av 63 JLabel-komponenter för att visa innehåll i *array7x7* och ytterligare några JLabel-komponenter som rubriker.

Fönstret **Array7x7 input** ger användaren möjlighet att ange värden i en rad eller kolumn vilken sedan ska lagras i *array7x7*. *array7x7* ska därefter visas i *Array7x7-viewer*. Användaren kan också ange en speciell rad eller kolumn i *array7x7* vilken ska visas i *Array7x7 viewer*. JPanel-komponenten i fönstret innehåller 16 JTextField-komponenter i vilka värden kan matas in, 4 JButton-komponenter och några JLabel-komponenter.



Column	Array7x7						
0	0	2	0	2	0	0	0
0	0	4	0	4	0	0	0
0	0	9	0	0	0	0	0
0	0	7	5	2	0	9	0
0	0	9	0	4	0	0	0
0	0	1	0	1	0	0	0
0	0	1	0	1	0	0	0
Row	0	0	0	0	0	0	0

Figur 4.1, Visare av innehåll i en *Array7x7*-instans



Array7x7 input	
Column nbr: 1	2
Read column	4
Write column	9
	7
Row nbr: 2	9
Read row	1
Write row	1
0 9 0 0 0 0 0	

Figur 4.2, Inmatning av värden mm

Vid klick på "Read row" ska värdena i angiven rad (efter Row nbr:) läsas från *array7x7*. Därefter ska den lästa raden visas i raden längst ner i *Array7x7 viewer*.

Vid klick på "Write row" ska värdena i raden längst ner skrivas till angiven rad i *array7x7*. Sedan ska innehållet i *array7x7* visas i *Array7x7 viewer*.

Vid klick på "Read column" ska värdena i angiven kolumn (efter Column nbr:) läsas från *array7x7*. Därefter ska den lästa kolumnen visas i kolumnen till vänster i *Array7x7 viewer*.

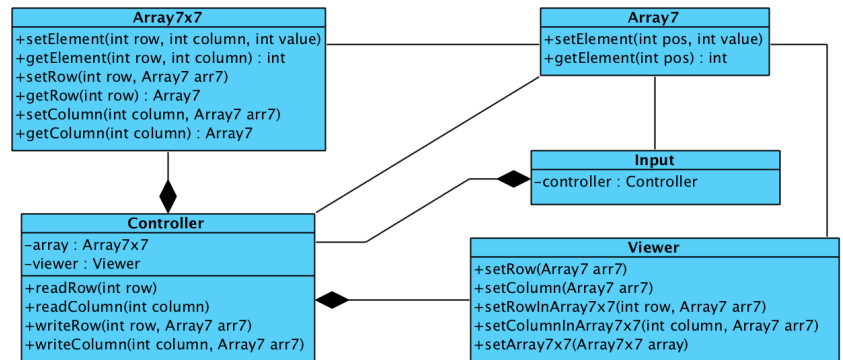
Vid klick på "Write column" ska värdena i kolumnen till höger skrivas till angiven kolumn i *array7x7*. Sedan ska innehållet i *array7x7* visas i *Array7x7 viewer*.



Klassdiagrammet visar designen av systemet. Klassdiagrammet visar *samtliga* public-deklarerade metoder och några väsentliga attribut i klasserna *Controller*, *Input* och *Viewer*. Klasserna *Input* och *Viewer* ärver av *JPanel* och är de paneler som visas i *Array7x7 input* respektive *Array7x7 viewer*.

Controller-instansen:

- Har ett attribut av typen *Array7x7* (*array7x7* nedan)
- Anropas av *Array7x7 input* då användaren klickar på en knapp. Eventuellt lagras värden i *array7x7* och slutligen visar någon form av resultat i *Array7x7 viewer*.



5 Moment 3: Utöka med metoder i *Array7x7* och testa dessa utförligt

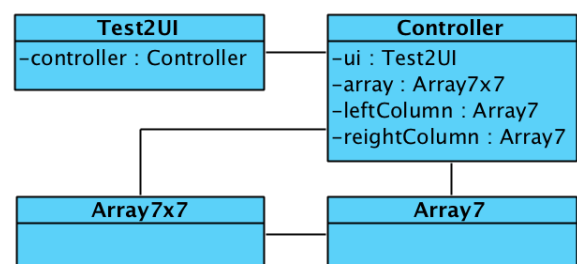
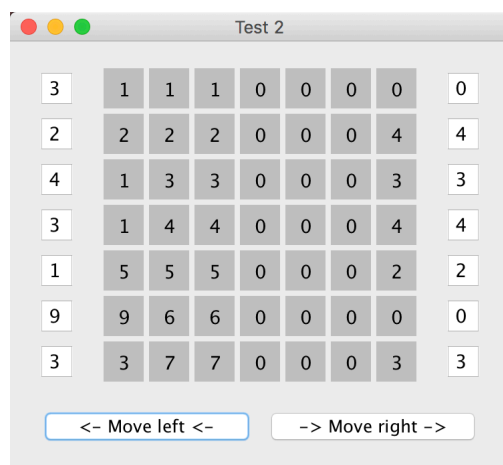
5.1 Uppgifter

Uppgift 5.1.1:

Array7x7 skall nu utökas med metoder för att skifta innehållet åt vänster, *shiftLeft*, och skifta innehållet åt höger, *shiftRight*. Ett tips är att göra rutinerna så att man skickar in en *Array7* (som kopieras in till höger då man skiftar vänster och till vänster då man skiftar höger) och får ut en *Array7* som innehåller det som låg i den kolumn som "trillade över kanten".

Uppgift 5.1.2

Ett java-program tas fram som använder klasserna som skapades i föregående moment. Kolumnerna till vänster och höger består av 7 st *JTextField*-komponenter vardera. I mitten är det 49 st *JLabel*-komponenter. Längst ner är det 2 *JButton*-komponenter.



Figur 5-1. Testuppställning för test av skift till vänster respektive höger och en tänkbar design av systemet

De 49 *JLabel*-komponenterna visar värdena som lagras i en *Array7x7*-instans, *array7x7* nedan.



Vid klick på "Move left" ska samtliga värden i `array7x7` flyttas en kolumn åt vänster genom att anrop av metoden `shiftLeft` i `array7x7`. Värdena i kolumnen till höger (JTextField-komponenter) ska skrivas till sista kolumnen i `array7x7`. Första kolumnens värden i `array7x7` ska visas i kolumnen längst till vänster (JTextField-komponenter).

I figuren ovan har "Move left" just klickats. Värdena i vänster kolumn (3, 2, 4, 3, 1, 9, 3) var tidigare längst till vänster bland JLabel-komponenterna (där 1, 2, 1, 1, 5, 9, 3 är nu)

Vi klick på "Move right" ska motsvarande beskrivningen ovan inträffa men allt ska flytta åt höger.

6 Moment 4: Använda Array7 och Array7x7 vid hantering av färgelement

6.1 Beskrivning

Klassen `Color` bifogas och ska användas för att hantera färgvärden i Eclipse (standard java). Klassen `Color` fungerar tillsammans med klassen `ColorDisplay`. I klassen `Color` representeras färg med 4 bytes i en `int`. De 4 byte-värdena representerar *alfa* (0-255 där 0 är helt transparent och 255 är helt tät), *röd* (0-255), *grön* (0-255) och *blå* (0-255). Klassen `Color` innehåller metoder för konvertering från färgkomponenter till färg-`int` och från färg-`int` till färgkomponenter. Dessutom innehåller klassen ett antal färgkonstanter. OBS! Klassen `java.awt.Color` ska EJ användas tillsammans med `ColorDisplay`.

I Android representeras färg på samma sätt som i klassen `Color` ovan. I android används därför standardklassen `Color` tillsammans med `ColorDisplay`.

I java respektive android bifogas klassen `ColorDisplay` vilken visar 49 färger i en 7x7 rutor stor display. I båda fallen representeras färgerna av `int`-värde vilka skapas genom `Color`-klassen.

Klasserna `Array7x7` respektive `Array7` kan lagra `int`-värden och därmed färgvärden.

6.2 Uppgifter

Uppgift 6.1.1:

Skapa ett `Array7x7`-objekt med färgvärden. Testa att visa objektet i en `ColorDisplay`, skifta sedan raderna åt vänster och visa objektet på nytt. Fortsätt och använd andra metoder i `Array7x7`-objektet och kontrollera resultatet av dem.

7 Moment 5: Designa och implementera alla bokstäver (A-Z, 0-9, etc.)

7.1 Uppgifter

Skapa `Array7x7`-objekt för alla tecken som ska kunna visas i displayen, dvs A-Z, 0-9 och vissa tecken som t.ex. `.,+-*@?()\ /<space>...` Tecknet för `<space>` är ganska enkelt att designa.

Skapa också ett tecken, som kan användas då en bokstav/tecken som inte stöds påträffas. Detta kan tex vara en helt fylld ruta.

Uppgift 7.1.1:

Skapa ovanstående tecken. Varje tecken definieras av en 7x7 array. Elementen i arrayen talar om hur varje ruta i arrayen ska visas, så att tecknet ser vettigt ut. När arrayen är korrekt så kan `Array7x7`-objekt skapas.



På något sätt måste tecken och objekt kopplas samman. Bilaga 2 ger förslag på ett par sätt att koppla samman tecken och objekt.

8 Moment 6: Implementera rinnande text och testa detta

8.1 Uppgifter

Uppgift 8.1.1:

Låt användaren mata in en text via ett inmatningsfönster. Använd sedan inmatningen för att visa tecknen, t.ex. ett tecken åt gången, i displayen med hjälp av en Timer (java.util.Timer).

Använd sedan en större display, t.ex. med bredden 35 rutor, och låt texten flöda över denna. Flytta varje tecken en kolumn åt vänster med visst intervall (använd timer)

9 Moment 7: Gör ett klassdiagram över lösningen i moment 6

9.1 Uppgifter

Uppgift 9.1.1:

Gör ett klassdiagram över lösningen i moment 6 (rinnande text). I klasserna ska det ej tas med metoder och endast viktiga attribut.

10 Moment 8: Flytta över till Android och testa

10.1 Beskrivning

Nu skall koden flyttas till Android och testas där.

10.2 Uppgifter

Uppgift 10.1.1:

Flytta klasserna (inte testmiljöerna) till Android. Det är UI-klasserna som måste skrivas för Android. Övriga delar av lösningen i Moment 6 går att återanvända.

11 Moment 9: Optionella uppgifter

Om tiden medger kan gruppen t.ex.:

- Låta texten flöda åt höger / uppåt / nedåt
- Visa rörlig text mot en bakgrundsbild
Java: Placera ColorDisplay-komponenten på en IconPanel (se Bilaga 3), sätt bakgrundsfärgen och gridfärgen Color.TRANSPARENT i Colordisplay-komponenten.
Android: Lägg till attributet android:background i layout-elementet som innehåller ColorDisplay-komponenten och sätt bakgrundfärg och gridfärg till Color.TRANSPARENT i ColorDisplay-komponenten.
- Gör ett kaleidoskop
- Visa en bild i bakgrunden och placera ColorDisplay-komponenter som täcker bakgrunden. Gör successivt rutor i ColorDisplay-komponenten transparenta. Vem ser först vad bilden föreställer?
- Ta en bild och visa den i en ColorDisplay. Använd ett lämpligt antal rutor på bredden och höjden och googla fram en algoritm för att representera ett antal pixlar i bilden som en ruta med färg.



Bilaga 1

Beskrivning av klasser vilka levereras vid projektstart. Ett exempel på hur man använder klasserna Color och ColorDisplay hittar du i javafiles.zip (main-metod i Controller).

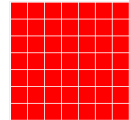
ColorDisplay

Konstruktörer

public ColorDisplay(int background, int grid)

Konstruerar 7x7 stor display med angiven bakgrundsfärg och färg på griden. Griden är två pixlar bred och sidan på varje ruta sätts till längden 40 pixlar.

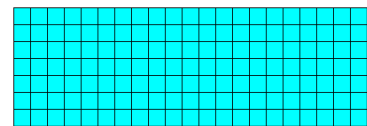
Ex: `new ColorDisplay(Color.RED, Color.WHITE)`



public ColorDisplay(int verticalPages, int horizontalPages, int background, int grid)

Konstruerar (verticalPages-7)x(horizontalPages-7) stor display. Griden är två pixlar bred och sidan på varje ruta sätts till längden 40 pixlar.

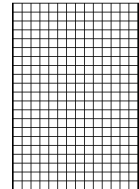
Ex: `new ColorDisplay(1, 3, Color.CYAN, Color.BLACK)`



public ColorDisplay(int verticalPages, int horizontalPages, int background, int grid, int gridStroke, int sideSize)

Konstruerar (verticalPages-7)x(horizontalPages-7) stor display. Griden är gridStroke pixlar bred och sidan på varje ruta sätts till längden sideSize pixlar.

Ex: `new ColorDisplay(3, 2, Color.WHITE, Color.BLACK, 1, 10)`



Metoder

public void setBackgroundColor(int background)

public void setGridColor(int grid)

public void setGridStroke(int gridStroke)

public void setSideSize(int sideSize)

public int getBackgroundColor() : int

public int getGridColor() : int

public int getGridStroke() : int

public int getSideSize() : int

public int getHorizontalPages() : int

public int getVerticalPages() : int

public void clearDisplay()

Displayens rutor ges bakgrundsfärg

public void setDisplay(int[][] colors)

Värdena i colors överförs till displayens 7 första rader och kolumner. Anrop till denna metod motsvara anrop till nedanstående setDisplay med argumenten verticalPage==0 och horizontalPage==0. colors ska ha dimensionen 7x7.

public void setDisplay(int[][] colors, int verticalPage, int horizontalPage)

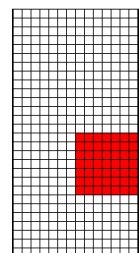
Värdena i colors överförs till raderna (verticalPage*7) – (verticalPage*7+6) och till kolumnerna (horizontalPage*7) – (horizontalPage*7+6) i displayen.

colors ska ha dimensionen 7x7.

Ex:

```
ColorDisplay d = new ColorDisplay(4, 2, Color.WHITE, Color.BLACK, 1, 10);  
d.setDisplay(arr, 2, 1); // arr har röd-värde i samtliga element  
d.updateDisplay();
```

ger displayen till höger



public void updateDisplay()

Displayens rutor uppdateras till senaste värden angivna med anrop till setDisplay.



IntDisplay

Konstruktörer

public IntDisplay(int background, int grid)

Konstruerar 7x7 stor display med angiven bakgrundsfärg och färg på griden.

Heltalen skrivs ut med svarta siffror

Ex: `new IntDisplay(Color.RED, Color.LTGRAY);`

public IntDisplay(int background, int grid, Mode mode, int textColor)

Konstruerar 7x7 stor display med angiven bakgrundsfärg och färg på griden.

Heltalen skrivs ut i angiven mode och med angiven färg. Displayen ovan till höger motsvarar:

`new IntDisplay(Color.RED, Color.LTGRAY, Mode.SHOW_INT, Color.BLACK);`

Displayen till höger motsvarar:

`new IntDisplay(Color.WHITE, Color.CYAN, Mode.SHOW_BYTE, Color.BLACK);`

Heltalen skrivs ut som färvärde med färgkomponenterna alfa, röd, grön, blå

Metoder

public void setTextColor(int textColor)

public void setBackground(int background)

public void setGrid(int grid)

public void setMode(Mode mode)

Mode kan vare något av värdena Mode.SHOW_INT och Mode.SHOW_BYTES

public int getBackgroundColor()

public int getGridColor()

public void clearDisplay()

Displayens värden nollställs

public void setDisplay(int[][] values)

Värdena i *values* överförs till displayen.

public void updateDisplay()

Displayens rutor uppdateras till senaste värden angivna med anrop till *setDisplay*.

-2158	50597	-3122	-3213	-8649	18853	18252
11239	2084	43420	38567	33511	31159	41652
50988	-7410	15129	-3444	46355	-9345	-7515
9728	83230	373	85402	3892	8112	4934
85898	-5375	-1380	-1784	8633	78891	1542
68812	53787	-68888	8888	-86888	7881	8587
-57623	-3741	-4140	87658	-1358	-5619	12301
7101	10100	73450	2731	43314	16749	8750
64895	-1025	21127	-1485	-7061	17729	87493
89385	84848	8938	188371	188378	8280	8812
-5482	-5428	88483	-7488	88484	32884	18831
59079	76300	7972	48471	17	12	85593
-6895	-1130	73820	-1808	-8755	-9071	83728
12952	-99438	4855	16887	184	17288	6589

53,151	40, 13	227,118	27,170	7,218	30,150	33,139
81, 41	123, 43	163,236	53,137	229,191	77, 50	206,168
35,238	214, 56	236, 24	209,152	212,126	6, 79	222,138
220, 1	55,108	236,240	219, 89	156,148	69,240	174, 40
235,129	243, 70	3,166	157, 21	37, 84	231,100	150,106
115,171	3, 9	226, 95	25,189	232,164	94,201	23,184
213,137	221,111	221,245	242, 67	250,176	3,163	206,177
248,168	64, 86	149, 37	90, 82	109,140	115, 62	97,124
135, 4	44,108	200, 25	3, 69	109,222	26,177	224,215
53, 99	62, 71	165,165	61, 59	14,176	48,226	102, 28
6,193	211, 97	58,150	25, 38	15,124	49, 59	59,116
200,108	222,135	167,221	18, 37	47,220	202, 69	253,144
225,147	6,109	36, 68	235, 29	43, 44	46, 48	14, 92
162,241	166,189	179,172	169, 84	69, 67	177, 75	229,175



Color

Innehåller klassmetoder för att skapa färgvärde:

public static int rgb(int red, int green, int blue) // 0-255 för samtliga värden

public static int argb(int alpha, int red, int green, int blue) // 0-255 för samtliga värden

public static int parseColor(String colorString)

colorString ska vara på något av formaten:

- "#RRGGBB" – RR (rödvärde), GG (grönvärde), BB (blåvärde). Värdena anges hexadecimalt.
- "#AARRGGBB" – AA (alfavärde), övrigt som ovan.

Innehåller klassmetoder för att få färgkomponent ur ett färgvärde (*color* nedan):

public static int alpha(int color)

public static int red(int color)

public static int green(int color)

public static int blue(int color)

Innehåller klassmetod för att skriva ut ett färgvärde (*color* nedan):

public static String toString(int color)

Innehåller ett antal färgkonstanter:

Color.BLACK, Color.BLUE, Color.CYAN, Color.DKGRAY, Color.GRAY, Color.GREEN, Color.LTGRAY,
Color.MAGENTA, Color.RED, Color.TRANSPARENT, Color.WHITE, Color.YELLOW



Bilaga 2

En klass, Chars, vilken kopplar samman tecken med Array7x7-objekt. Två alternativ ges nedan.

Båda alternativen kräver att varje tecken definieras, t.ex. med en tvådimensionella array:

```
private static int[][] charA = {  
    {0,0,0,1,0,0,0},  
    {0,0,1,0,1,0,0},  
    {0,0,1,0,1,0,0},  
    {0,0,1,1,1,0,0},  
    {0,1,0,0,0,1,0},  
    {0,1,0,0,0,1,0},  
    {0,1,0,0,0,1,0}};
```

```
private static int[][] SPACE = {  
    {0,0,0,0,0,0,0},  
    {0,0,0,0,0,0,0},  
    {0,0,0,0,0,0,0},  
    {0,0,0,0,0,0,0},  
    {0,0,0,0,0,0,0},  
    {0,0,0,0,0,0,0},  
    {0,0,0,0,0,0,0}};
```

```
private static int[][] DOT = {  
    {0,0,0,0,0,0,0},  
    {0,0,0,0,0,0,0},  
    {0,0,0,0,0,0,0},  
    {0,0,0,0,0,0,0},  
    {0,0,0,0,0,0,0},  
    {0,0,0,0,0,0,0},  
    {0,0,0,1,0,0,0}};
```

```
private static int[][] UNKNOWN = {  
    {0,1,1,1,1,1,0},  
    {0,1,0,0,0,1,0},  
    {0,1,0,0,0,1,0},  
    {0,1,0,0,0,1,0},  
    {0,1,0,0,0,1,0},  
    {0,1,0,0,0,1,0},  
    {0,1,1,1,1,1,0}};
```

Alternativ 1

Sedan kan man skriva en metod vilken returnerar ett Array7x7-objekt för givet tecken, ungefär:

```
public static Array7x7 getChar(char chr) {  
    Array7x7 res;  
    switch(chr) {  
        case 'A' : res = new Array7x7(charA); break;  
        // övriga tecken, det blir en lång metod  
        default : res = ...  
    }  
    return res;  
}
```

Med denna lösning skapas ett nytt Array7x7-objekt vid varje anrop till getChar-metoden.



Alternativ 2

Låt klassen ha instansvariabeln:

```
private Array7x7[] chars; // med kapaciteten 128
```

I konstruktorn kan arrayen instansieras med lämplig kapacitet. Sedan kan det vara en god ide att initiera samtliga element i arrayen med ett Array7x7-objekt som innehåller "okänt tecken"-tecken. Därefter kopplas tecken och Array7x7-objekt ihop.

```
chars['A'] = new Array7x7(charA);
```

```
chars['. '] = new Array7x7(DOT);
```

osv. en rad för varje definierat tecken.

Slutligen krävs det en metod, getChar(char chr), vilken returnerar referens till korrekt Array7x7-objekt avseende parametern av typen char.

Om chr>=0 och chr <=127 så // det går att göra så här i java, char lagras som int

```
returnera chars[chr];
```

annars

```
returnera "okänt tecken"-tecken
```

Med detta alternativ kommer varje tecken att representeras av ett objekt. Detta kan även gälla "okänt tecken"-tecknet. Detta tecken kan återanvändas om det är instansvariabel i klassen.



Bilaga 3

Klassen IconPanel

```
package project;

import java.awt.Dimension;
import java.awt.Graphics;
import javax.swing.Icon;
import javax.swing.JPanel;

public class IconPanel extends JPanel {
    private Icon icon;

    public IconPanel(Icon icon) {
        this.icon = icon;
        this.setPreferredSize(new Dimension( icon.getIconWidth(),
                                              icon.getIconHeight()));
    }

    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        icon.paintIcon(this, g, 0, 0);
    }
}
```