06-Nov-2012/FK

ID1004 Laboration 4, 14-16 November 2012

Beräknad tid ca 1-2 timmar.

Instruktionen antar att labben utförs i datasal, med hjälp av den integrerade utvecklingsmiljön Eclipse. Alternativt kan du använda egen dator, någon lämplig texteditor, Java JDK och kompilera och köra själv. Om du väljer editorn TextPad, tänk att ha installerat Java JDK innan du installerar TextPad, eftersom TextPad vid sin installation då konfigureras för att stödja källkod i Java.

Labben innehåller uppgifter av olika karaktär och svårighetsgrad. Försök att bli helt klar med en lättare uppgift innan du ger dig på en av de svårare.

Resurser online

Java version 6 API dokumentation

http://download.oracle.com/javase/6/docs/api/

Java JDK download (SE version 6)

http://www.oracle.com/technetwork/java/javase/downloads/index.html

TextPad editor for Windows

http://www.textpad.com/

Eclipse IDE (välj Classic)

http://www.eclipse.org/downloads/

Uppgift L4.1 Modellering av rollspelstärningar (lätt)

Den här uppgiften går ut på att modellera rollspelstärningar. I rollspel med tärningar används vanliga sexsidiga tärningar men också ofta tärningar med 4, 6, 8, 10, 12, 16, eller 20 sidor.

Superklassen Tärning

Skapa först en klass Tärning (*utan* metoden main). Låt klass Tärning ha en variabel

```
private final int antalSidor;
```

som sätts av konstruktorn:

```
public Tärning(int sidor)
```

Låt klass Tärning ha en metod

```
public int kasta()
```

som returnerar ett heltal mellan 1 och det antal sidor som tärningen har. Ett sådant slumptal kan genereras från Math.random() på detta sätt:

1 + (int) (Math.random() * antalSidor);

Skriv metoden

```
public String toString()
```

så att antalet sidor syns i den returnerade strängen. Till exempel: `”Tärning 12”`

Subklasserna T4 – T20

Skapa därefter tre subklasser (*utan* metoden main) för tärningar med olika sidor. Välj mellan T4 (fyra sidor), T6 (sex sidor), T8, T10, T12, T16 och T20. Låt dina klasser ärva (extends) från Tärning.

I var och en av de tre subklasser du väljer, skapa en konstruktor som i sin tur anropar konstruktorn i Tärning för att sätta antalet sidor som just den klassen ska ha. Till exempel för en tärning med 16 sidor:

```
public T16 () {
  super(16);
}
```

Eftersom ingen av klasserna Tärning eller T*n* ska innehålla någon metod main, skriv ett separat testprogram som instantierar dina valda tärningar och lägger dem i en array av Tärning, t ex:

```
Tärning [] minaTärningar = {new T4(), new  T12(), newT20()};
```

Skriv sedan kod som går igenom arrayen med en for-each loop och kastar varje tärning tre gånger. I varje kast, skriv ut vad det är för tärning och vad kastet blev.

Uppgift L4.2 Behållare (medium)

Modellera och implementera behållare. En behållare har en storlek (dvs kräver ett visst utrymme) (ett tal), och har en viss kapacitet (dvs hur många storlekar den rymmer) (ett tal). Varje behållare ska ha en etikett i form av en sträng så att den går att identifiera, t ex ”blå kartong” eller ”plastask”.

Etikett, storlek och kapacitet ska gå att sätta med konstruktorn. Om kapaciteten överskrider storleken ska en IllegalArgumentException kastas från konstruktorn. En exception är också ett objekt och kan skapas och kastas så här:

```
throw new IllegalArgumentException(”Capacity exceeds size for ” + label);
```

Det innebär att konstruktorn får denna deklaration:

```
public Behållare(String label, int size, int capacity) throws
IllegalArgumentException
{
  …
}
```

Klassen Behållare ska även ha:

- En metod **`protected int sizeOfContents()`** som returnerar summan av innehållets storlek.
- En metod **`public boolean add(Behållare)`** som lägger en behållare i en annan, om den får plats, dvs om den återstående kapaciteten räcker till . Om den fick plats returneras true annars false. Använd gärna en **`ArrayList<Behållare>`** för att hålla reda på innehållet.
- En metod **`public String toString(String prefix)`** som returnerar en lista över behållarens innehåll. Listan kan byggas upp med hjälp av en **`StringBuffer`**. Första raden i listan innehåller prefix och behållarens etikett. Om behållarens inte är tom, så lägg även till texten `”innehåller”` till anropa därefter **`b.toString(prefix + ”  ”)`** för varje behållare b i innehållet. De returnerade strängarna läggs till i den returnerade strängen.
- En metod **`public String toString()`** som returnerar **`toString(””)`**. Resultatet kan då se ut så här:

```
koffert innehåller:
  ryggsäck innehåller:
    vit ask
    blå ask
    grön ask
  skokartong
```

Skriv ett separat testprogram som modellerar tre askar, en vit en blå och en grön. Lägg de tre askarna i en ryggsäck. Lägg ryggsäcken i en koffert. Lägg också en tom skokartong i kofferten. Skriv ut innehållet i kofferten och kontrollera att allting syns, ända ner till askarna.

Uppgift L4.3 Designa digitala filter (svår)

Skapa ett nytt projekt i Eclipse.

Hämta filen Filter.java från kth/social och lägg in den i projektet. Studera speciellt metoderna setFilter() och setFilterCharacter().

Titta på filterdefinitionerna nedan och skapa sedan en ny klass (utan main) som ärver från klass Filter; till exempel LowPassFilter, HighPassFilter, eller BandPassFilter. Låt klassen åsidosätta metod setFilterCharacter() med motsvarande beräkning.

Skapa sedan ett testprogram (en ny klass med main) som

- instansierar det skapade filtret,
- sätter lämpliga parametrar med setFilter()
- skickar in en array med 1000 samples där första elementet är 1 och resten 0 (impuls)
- skriver ut de 50 första samplen i impulssvaret.

Namnen cos, sin och alpha i filterberäkningarna nedan refererar till variablerna i klassen Filter.

Low pass filter

b0 = (1 – cos) / 2
b1 = 1 – cos
b2 = (1 – cos) / 2
a0 = 1 + alpha
a1 = -2 * cos
a2 = 1 – alpha

High pass filter

b0 = (1 + cos)/2
b1 = -(1 + cos)
b2 = (1+ cos)/2
a0 = 1 + alpha
a1 = -2 * cos
a2 = 1 – alpha

Band pass filter

b0 = sin / 2
b1 = 0
b2 = -sin / 2
a0 = 1 + alpha
a1 = -2 *cos
a2 = 1 – alpha

Metoden setFilter

Parametrarna till setFilter kan inte vara tagna ur luften:

Filterfrekvensen (där det händer) fqHz får inte vara större än sampleRate/2

Q-värdet bör hållas i området 0.2 – 3

Parametern gaindb används inte och kan sättas till 0

Exempel

Ett low pass (lågpass) filter definierat som

```
f.setFilter(8000d, 1000d, 0.5d, 0d);
```

ger följande impulssvar

```
0.08578644
0.24264069
0.27207792
0.18376616
0.10555567
0.055915885
0.028211748
0.013777727
0.0065734712
0.0030817576
0.0014251819
6.5191323E-4
2.9553997E-4
1.3298269E-4
5.9459817E-5
2.6441896E-5
1.1703491E-5
5.1587763E-6
2.2656682E-6
9.918349E-7
4.3293565E-7
1.8848365E-7
8.1864954E-8
3.548046E-8
1.5347167E-8
6.6265233E-9
2.8564338E-9
```

Mer information för den intresserade kan hittas bl a här:

http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt