Ursprungligen skriven av Erland Holmström.
Magnus Myreen har uppdaterat vissa delar.
Hösten 2014

Laboration 3, uppgift 1

3.1 En klass för en räknare

Anvisningar:

Programmet skall utformas enligt de principer som lärts ut i kursen och koden skall vara indenterad mm enligt stilguiden på hemsidan, se länksidan.

För att bli godkänd räcker det alltså inte med att programmet fungerar korrekt, det måste vara välstrukturerat och läsbart också!

Nu börjar vi också tycka att ni skall kunna saker som att lämna in korrekt och att indentera mm så i fortsättningen kommer vi att bedöma sådant hårdare.

Tänk också på att om du skall skriva ett testprogram så skall du göra det och testprogrammet skall **testa** dina klasser/metoder. Det är inte frågan om att slänga in ett enkelt test och sedan dra vidare till mitt testprogram (tex att testa en eller ett fåtal matris(er) i Rse) utan du skall skriva ett testprogram som gör sitt jobb dvs försöker avgöra om ditt program fungerar.

Förkunskaper:

Den här uppgiften kräver kunskap om klasser, arv, abstrakta klasser och interface.
Efter föreläsning 10 har vi täckt detta.

Syfte:

Att öva på: metoder, klasser, instans-klassvariabler, arv, abstrakta klasser, interface, fält, att testa program.

Huvudsyftet här är att hantera enkla klasser på olika sätt så uppgiften består av flera små uppgifter.

Inlämning:

Du redovisar java klasser enligt:

`CounterModel.java` innehåller lösning på räknaren som den ser ut efter **Del (e)**.

`TestCounter.java` innehåller testprogramet efter **Del (g)** (som innehåller alla tester så långt). Testerna är naturligtvis anpassade efter hur räknaren ser ut efter **Del (g)**.

`FastCounter.java` innehåller lösning på **Del (h)**.

`ChainedCounterModel.java` innehåller lösning på **Del (i)**.

och i en separat mapp som du döper till ”javadoc”, finns Javadoc filerna för samtliga klasser ovan.

Dessutom så skall frågorna (märkta med ”*” i början på frågan) besvaras i `README` filen. Numrera svaren med samma bokstav som uppgiften och upprepa frågan först i ditt svar. Här finns många frågor som inte behöver redovisas. Men dom kommer kanske på tentan!

OBS: Labben består av två uppgifter. Den andra delen handlar om rekursion som vi går (eller gick) igenom i föreläsning 11.

Uppgiften:

Antag att du behöver en klass för en enkel räknare. Man skall kunna öka/minska räknaren.

Del (a)

Du skall göra en klass för en sådan räknare. Kalla den `CounterModel`.
Du behöver inte göra någon konstruktor än - hur konstrueras objekt då?

Specifikationen för räknaren, utan semantik ännu eftersom ni skall skriva den själva sedan i era klasser, ser ut så här (och den skall ditt program implementera)

```
public interface CounterInterface {
public void increment();
public void decrement();
public void reset();
public int getValue();
} // end CounterInterface
```

`increment` ökar med ett, `decrement` minskar med ett, `reset` nollställer och `getValue` returnerar värdet. Räknaren skapas med värdet noll. Tänk på att alla instansvariabler skall vara `private`!

Skriv också ett program som testar din räknare (separat klass i en separat fil med namn `Test-Counter`). Du måste skapa mer än en räknare i testprogrammet (med olika namn) och måste använda dem oberoende av varandra. Gör lämpliga utskrifter.
Tänk igenom vilka tester du skall genomföra. Titta på hur testprogrammen såg ut i lab 2 och läs nedan.

Del (b)

Vilken utskrift fås när du kör följande program? Lös det "på papper" först innan du provkör.

* Förklara varför varje rad av utskriften ser ut som den gör, rita gärna.

```
public class VadSkrivsUt {
  public static void changeValues(int n,
                                  CounterModel c1,
                                  CounterModel c2) {
    n++;
    c1.increment();
    c2 = new CounterModel();
```

```
    c2.increment();
  }
  public static void main(String[] args) {
    int k=5;
    CounterModel a = new CounterModel();
    CounterModel b = new CounterModel();
    changeValues(k,a,b);
    System.out.println("k="+k);
    System.out.println("a="+a.getValue());
    System.out.println("b="+b.getValue());
  } // end main
} // end VadSkrivsUt
```

Vi skall nu lägga till lite funktioner till vår räknare.

Del (c)

* Vad händer om man gör om instansvariabeln i CounterModel som håller reda på räknaren till "*static*"? (Om det "fungerar" ändå så har du för snällt testprogram!) Förklara skillnaden i beteende om man använder en instansvariabel respektive klassvariabel.

Lägg till en metod getNbrOfCounters (och en variabel) i CounterModel som kan returnera hur många räknare som skapats. * Skall denna metod vara en klassmetod eller en instansmetod? * Kan den vara bägge? * Vilket är lämpligast? Du behöver nog en (parameterlös) konstruktor nu.

Del (d)

Det är ju lite tråkigt med textutskrifter så därför har du tänkt presentera din räknare enligt figuren till höger. De fyra filerna i mappen `simpleCounter` som finns på hemsidan bildar tillsammans med din class `CounterModel` ett fullständigt grafiskt program som du skall provköra. Lägg alltså till din `CounterModel` klass i mappen `simple-Counter`, kompilera och provkör.

Funkar det bra? Om nån vecka eller två så kan du själv skriva ett sånt här program så du kan redan nu kika lite på hur det ser ut om du vill.

Del (e)

När du kör programmet ovan så inser du (kanske) att räknaren måste kunna begränsas till att räkna mellan 0..99 annars kommer den tex att visa "1" för "101" osv.

Gör alltså en konstruktor till klassen som har en parameter som säger hur långt räknaren skall kunna räkna. Den skall även göra sk. "wrap around" dvs att när räknare når sitt max värde så

skall den bli noll och när den blir mindre än noll skall den få sitt maxvärde. Konstruktorn tar en parameter `int modulus`, denna parameter anger vid vilket värde räknaren ska slå om till noll.

Om modulus är 60 och räknaren är "58" och man gör 2 "`increment`" så skall räknaren vara 0. Gör man nu 1 "`increment`" till så skall den bli 1. Är räknaren 3 och man gör 5 "`decrement`" skall den bli 58.

Använd modulus operatorn (`%`) för att öka räknaren i metoden `increment` (ingen if sats alltså) och en if sats för att minska den i `decrement` (bara för att öva på olika metoder att göra det).

Du behöver också en "getter" för `modulus`.

Här finns en del detaljfrågor att fundera över. Hur skall du hantera att konstruktorn får en negativ parameter? För en icke grafisk räknare så är det kanske inget problem men vår grafiska räknare klarar det inte (än i alla fall). Här är det nog bra med en exception tex IllegalArgumentException.

Samma `CounterInterface` och samma testprogram som tidigare kan användas med små förändringar - du skall ju tex använda den nya konstruktorn i testprogrammet. Den parameterlösa konstruktorn skall sätta modulus till 10 (som naturligtvis är en konstant).

Skriv också en `toString` metod och en `equals` metod. Tänk på att bägge dessa skall skrivas på speciella sätt. * Vad innebär det att två räknare är lika? Glöm inte komplettera testprogrammet med tester av dessa. Tänk på att både testprogrammets kod och dess utskrifter måste vara väl-strukturerade och lättlästa. (Om du inte vill köra alla tester varje gång så kan du ju kommentera bort de du vet fungerar eller bara göra utskrifter för fel svar.)

Nu är `CounterModel` klassen klar och du skall använda den på lite olika sätt *utan* att ändra den. Den här versionen av `CounterModel` skall också lämnas in.

Del (f)

Komplettera testprogrammet med kod som skapar *ett fält* av counters och ändrar/skriver ut de olika räknarnas värden. (Lägg detta efter testerna)

Del (g)

Komplettera testprogrammet med en metod som har en räknare, dvs en `CounterModel,` som parameter och som ändrar räknarens värde.

Provkör som vanligt. Du skall göra nåt i stil med

- skriv räknarens värde tex anropa `toString`
- anropa metoden som ändrar räknaren
- skriv räknarens värde

(allra bäst är det såklart om programmet själv kollar att svaren är korrekta.)

Del (h)

Skriv nu en klass, `FastCounter`, som ärver CounterModel och som utökar dess beteende med möjligheten att stega upp/ner med `x` steg i ett anrop dvs tex

`public void upMany()` och `downMany()`. Värdet `x` anges lämpligen till konstruktorn. Observera att det inte finns någon setter i `CounterModel` och du får inte ändra `CounterModel`.

* Vad händer om man anropar increment på en FastCounter?

Naturligtvis skall det finnas en `toString` och en `equals` även här.
Och en getter för "`x`" dvs `getStep()`.
Använd `modulus=15` och `step=5` som defaultvärden.

Del (i)

Nu vill vi ha ett program som fungerar som en klocka enligt figur nedan.

Det mesta finns i mappen `clockCounter` på hemsidan men klassen `ChainedCounterModel` saknas så den måste du skriva och lägga in i mappen innan du kan köra.

Notera att `CounterView` i denna mapp är samma klass som i föregående program; vi har här tre objekt av denna klass för vi har tre separata räknare. Och `CounterModel` behöver du också men den skall du inte behöva ändra heller.

Programmet startas genom att man ger aktuell tid på kommandoraden, t ex

```
> java Main 8 42 36
```

Klassen `ChainedCounterModel`, som du alltså måste skriva, skiljer sig från `CounterModel` på följande sätt:

Konstrueraren tar tre parametrar:

- `int init`, räknarens startvärde.
- `int modulus`, det tal som vi räknar modulo.
- `CounterInterface next`, en annan räknare. Se nedan för hur den används.

Metoden `increment` fungerar också annorlunda. Förutom beteendet tidigare gör man här följande: När räknaren slår om till noll för att värdet annars skulle bli `modulus`, så anropas `next.increment()`. Det är precis så en (digital) klocka fungerar; när t ex sekundräknaren når 60, så blir den i stället noll och minuträknaren ökas med ett. Man måste kolla om `next` är `null` innan man försöker göra `next.increment()`. Anledningen är att en kedja av räknare

måste ta slut, i klockan har timräknaren ingen nästa räknare, så den konstrueras med `null` som tredje parameter. Du skall naturligtvis ärva klassen `CounterModel` men får inte ändra i den.

Hur testar du nu?

Phuuuu nu är uppgiften slutligen slut... Bara att gå vidare till nästa uppgift :-)

Lite allmänt om att testa.

Ett tests uppgift är att jämföra faktiskt resultat med vad man förväntas få. Det är oftast inte möjligt att testa alla möjliga vägar som programmet kan gå men man får försöka tänka efter och hitta någorlunda relevanta tester. Speciellt viktigt är att testa "randvillkor" dvs i räknarens fall när den passerar över sina gränser (om 10 till 11 fungerar så fungerar nog 11 till 12 också men när räknaren skall gå från sitt maxvärde till noll kan det gå fel även om 10 till 11 fungerar)

Test bör, rent allmänt, antingen lyckas eller misslyckas. Man bör skriva ut

- vad man testar,
- vad man fick för resultat,
- vad man förväntades få
- samt om testet gick fel och isåfall kanske en markör typ "`*********`"

det sista så man dels slipper sitta och kolla alla värden varje gång och så man direkt ser om det är fel och kan undersöka vad som är fel, det skall inte kräva någon djup eftertanke att förstå utskriften. Testen bör även vara korta och självstående. Ibland nöjer man sig med att skriva ut det som blir fel men skriver inget om det är rätt, speciellt vid långa test.

Ett tänkbar sekvens när man testar räknaren är t.ex. för att kontrollera att räknaren beter sig som väntat om man går under 0...

```
CounterInterface c = new CounterModel(max);
c.decrement();
//
int result = c.getValue();
System.out.print("Test: down once." +
            " Expected: " + max-1 +
            " Actual:   " + result);
if ( result == max-1 ) {
    System.out.println(" Test successful");
} else {
    System.out.println(" --- TEST FAILED ---")
}
```

Med fördel lägger man "rätt"-testen (dvs efter kommentaren ovan) i en metod till vilken man skickar med det tänkta resultatet. Min metod har tex signaturen

```
private static void verify (
         CounterModel c,
         int value, int modulus, int nbrOfCounters, // bör-värden
         String message) {
         // message should be == "test x" where x is test number
```

I praktiken skriver man ofta test först utifrån vad man vill att sina metoder skall göra och skriver kod först efter att man har en testsvit.

Se också testprogrammen du fick på lab 2.