# DUGGA: Objektorienterade applikationer

## Läs detta!

- *Uppgifterna är inte avsiktligt ordnade efter svårighetsgrad.*
- **Skriv rent dina svar. Oläsliga svar r ä t t a s e j!**
- Programmen skall skrivas i Java 5, eller senare version, vara indenterade och renskrivna och i övrigt vara utformade enligt de principer som lärts ut i kursen.
- Onödigt komplicerade lösningar ger poängavdrag.
- Givna deklarationer, parameterlistor, etc. får ej ändras, såvida inte annat framgår av uppgiften.
- Läs igenom tesen och förbered ev. frågor.

| |
|---|
| I en uppgift som består av flera delar får du använda dig av funktioner klasser etc. från tidigare deluppgifter, även om du inte löst dessa. |

*Lycka till!*

## Uppgift 1 Modellering och designmönster

a) Komplettera följande klass så att designmönstret Singleton tillämpas:

```
public class MyClass {
   private Set<String> set;
   public MyClass() {
      set = new HashSet<>();
   }
   public void add(String s) {
      set.add(s);
   }
   public boolean contains(String s) {
      return set.contains(s);
   }
}
```

(3 p)

b) Matcha de tre diagrammen med de tre kodavsnitt som bäst motsvarar relationerna som uttrycks av pilarna.

Diagram 1

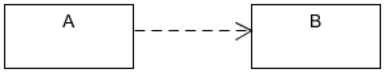
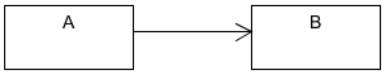



Diagram 2


Diagram 3




Kod 1

```
class A {
   B b;
   A(B b) {
      this.b = b;
   }
   ...
}
```

Kod 2

```
class A implements B {
   ...
}
```

Kod 3

```
class A {
   void f(B b){...}
   ...
}
```

Kod 4

```
class A extends B {
   ...
}
```

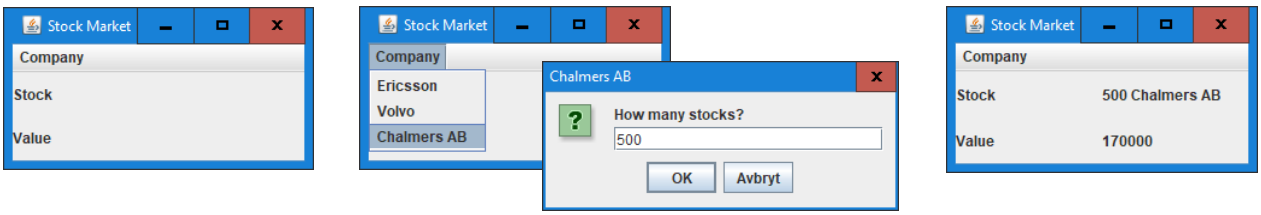
Kod 5

```
class A {
   B b = new B();
   ...
}
```

(3 p)

## Grafiska gränssnitt och MVC-modellen

En applikation för hantering av aktier (*eng.* stock = aktie) visar värdet för aktieposter i olika bolag. Man kan välja bolag i en meny och ange antalet aktier varefter värdet på aktieposten presenteras i fönstret. Nedanstående skärmbilder visar hur det skall se ut:




I uppgifterna 2 och 3 skall du implementera applikationen, dels ett grafiskt gränssnitt och en beräkningsdel, dels en kommunikationsdel som hämtar aktiedata från en server. Servern är given och skall ej konstrueras här. Applikation skall struktureras enligt designmönstret MVC. I båda uppgifterna används gränssnittet:

```
public interface DataSource {
   int getValue(String key) throws IOException;
}
```

Metoden `getValue` skall returnera ett värde för angiven nyckel, t.ex. `"Ericsson"`.

### Uppgift 2

a) Konstruera klassen `Gui`. Fönstret skall ha en meny enligt exemplet ovan, samt fyra textetiketter (JLabel) där information kan visas. Eventuella undantag i samband med kommunikation skall hanteras av GUI:t som visar ev. felmeddelanden i en pop-up-dialog. Klassen skall vara en observatör enligt designmönstret Observer. Dela upp koden i flera metoder.

(5 p)

b) Konstruera klassen `Model`. Klassen skall vara observerbar enligt designmönstret Observer. Klassen skall ha en publik metod

```
public void computeStockValue(String company,int noOfStocks)
throws IOException
```

som givet namnet på ett bolag och antalet aktier i bolaget beräknar värdet på aktieposten. Därefter uppdateras observatörerna. Du kan tills vidare anta att det finns en färdig klass

```
public class DatagramReceiver implements DataSource
```

för att hämta aktiekurser för olika bolag. (Denna klass konstrueras i nästa uppgift.)

(1 p)

c) Konstruera klassen `Main`. Klassen skall en `main`-metod som skapar och kopplar ihop objekt av klasserna `Model` och `Gui` till ett fungerande program.

(1 p)

### Uppgift 3

Konstruera klassen `DatagramReceiver`. Klassen skall implementera gränssnittet `DataSource` ovan. Metoden `getValue` skall hämta aktiekursen för angivet bolag och returnera värdet. Protokollet för kommunikationen skall vara att först skicka ett datagram med innehåll på formen "REQUEST:*bolag*", t.ex. "REQUEST:Volvo" till adressen 123.4.5.6 och port 1234 och därefter ta emot ett datagram innehållande aktiekursen på samma port som användes vid sändningen. Om datagrammet sänds genom socketen `s` kan porten erhållas med anropet `s.getLocalhost()`.

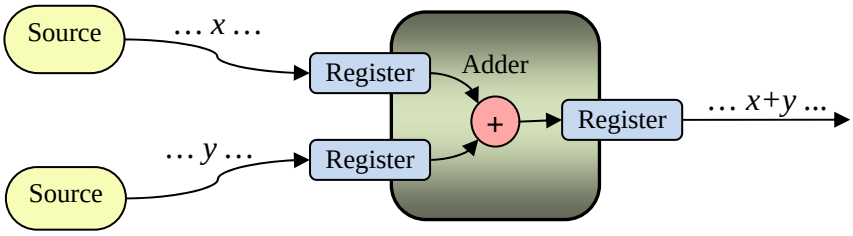
(5 p)

## Uppgift 4 Aktiva objekt och trådar

I denna uppgift skall vi göra en enkel modell av *register* och *adderare*. Ett register kan innehålla ett heltal. En adderare kan lägga ihop två heltal som hämtas från två inregister och placera summan i sitt utregister. Adderaren exekveras i en tråd och utför sin uppgift i oändlighet (i princip). För att inga tal skall tappas bort är det viktigt att användningen av registren synkroniseras. I figuren nedan syns även två objekt av (den färdiga) trådklassen:

```
public class Source {
    public Source(Register r) { ... }
}
```

Ett objekt av klassen placerar oupphörligt ett slumpmässigt heltal i registret som den är kopplad till. Genom att koppla ihop objekt av typen `Source`, `Register` och `Adder` (se nedan) kan vi skapa nätverk som gör beräkningar på dataströmmar, t.ex.




a) Konstruera klassen

```
public class Register {
    public void load(int value) { ... }
    public int read() { ... }
}
```

Observera att det endast skall finnas plats för ett heltal åt gången. Om metoden `load` anropas från en tråd skall talet lagras om registret är tomt, annars får tråden vänta. Om metoden `read` anropas från en tråd när registret är fullt tas talet bort och returneras, annars får tråden vänta. Utnyttja javas möjligheter till metodsynkronisering.

(4 p)

b) Konstruera klassen

```
public class Adder
{
    public Adder(Register in1,Register in2,Register out) {...}
}
```

Varje objekt av klassen skall exekvera en egen tråd och oupphörligt hämta heltal från sina inregister och ladda summan av talen i utregistret.

(1 p)

c) Skriv en `main`-metod som skapar och kopplar ihop objekt som realiserar figuren ovan. Lämpliga trådar skall startas och de producerade talen skrivas ut i en oändlig loop.

(1 p)

## Uppgift 5 Strömmar och filer

I en binärfil med tal av typen **`int`** lagras temperaturvärden från en väderstation. Vi är intresserade av att titta på extremvärden i filen, d.v.s. låga och höga temperaturer. Skriv ett program som läser en sådan binärfil, och skriver ut extremvärdena i en textfil. Textfilens namn skall vara samma som binärfilens, med tillägget ".txt". Eventuella undantag skall leda till lämpliga utskrifter. Namnet på indatafilen och temperaturgränserna ges som programargument:

```
> java ExtremeFinder temperatures -30 40
```

```
public class ExtremeFinder {
    public static void main(String[] arg) {
        ...
    }
    ...
}
```

Innehållet i utdatafilen `temperatures.txt` skall t.ex. kunna se ut så här:

```
Temperatures below -30:
-46 -44 -39 -50 -46 -45 -36 -50 -31 -33 -40 -45 -38 -31 -39
Temperatures above 40:
41 43 47 49 49 43 45
```

Notera att alla låga extremvärden skall skrivas ut först och de höga sist.

(6 p)