TENTAMEN: Objektorienterade applikationer

Läs detta!

- *Uppgifterna är inte avsiktligt ordnade efter svårighetsgrad.*
- Börja varje uppgift på ett nytt blad.
- Skriv ditt idnummer på varje blad (så att vi inte slarvar bort dem).
- **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, renskrivna och i övrigt vara utformade enligt de principer som lärts ut i kursen.
- Onödigt komplicerade lösningar ger poängavdrag.
- Programkod som finns i tentamenstesen behöver ej upprepas.
- Givna deklarationer, parameterlistor, etc. får ej ändras, såvida inte annat sägs i uppgiften.
- Läs igenom tentamenstesen och förbered ev. frågor.

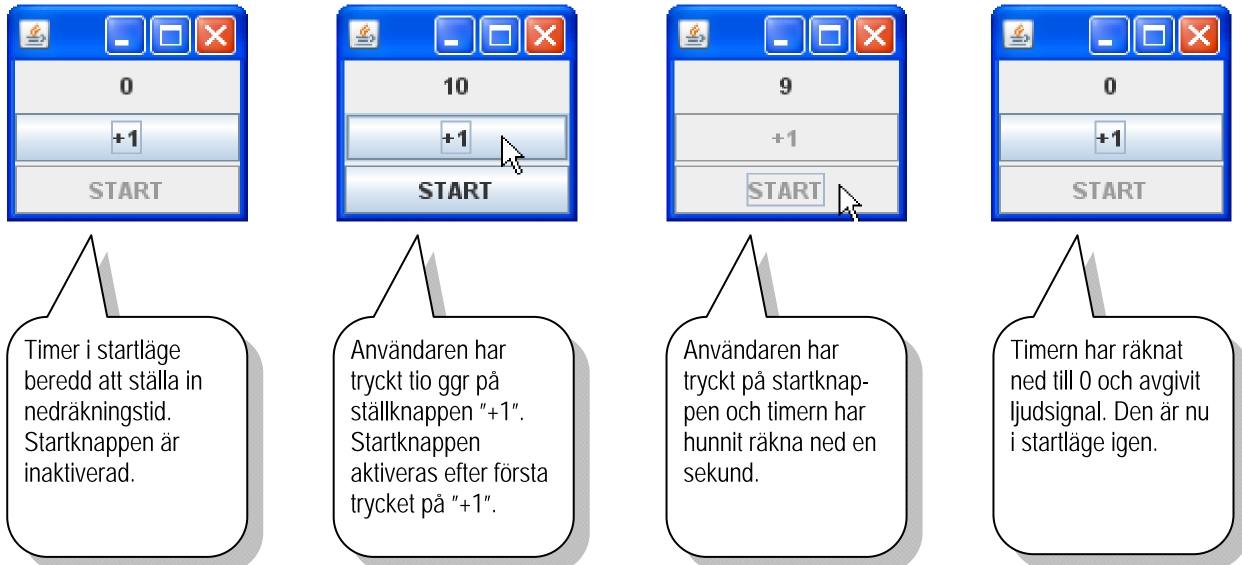
I en uppgift som består av flera delar får du använda dig av funktioner klasser etc. från tidigare deluppgifter, även om du inte löst dessa.

Tentamenskod:

Lycka till!

Uppgift 1

Vi skall göra en nedräkningstimer med ett grafiskt användargränssnitt. Den skall fungera så här:



GUI:t har tre komponenter, överst en `JLabel` där tiden visas, och under den två knappar. Om knapparna är aktiverade eller ej beror på timerns tillstånd enligt exemplen ovan. Tiden skall räknas upp med en enhet i det översta fönstret vid varje tryck på "+1", och ned med en enhet per sekund när timern är i nedräkningsläge tills den når 0.

a) Komplettera kodskelettet på följande sidor med
- ytterligare nödvändiga instansvariabler,
- lämplig konstruktor,
- kod som gör att fönstret skapas,
- ett timer-objekt av lämplig typ.

(6 p)

b) Inför händelsehantering. Bestäm själv om den skall vara centraliserad, eller med inre anonyma lyssnarklasser.

(6 p)

Några tips:
- Ljud: `Toolkit.getDefaultToolkit().beep();`

forts.

```
public class TimerGui
{
    private JLabel timeLabel;
    private JButton setButton, startButton;







    TimerGui() {









    }
```

```
} // End TimerGui
```

Uppgift 2

Du har fått i uppdrag att konstruera klassen `FileUtilities` med diverse filhanteringsmetoder. Bland annat skall klassen innehålla en metod för att översätta binära datafiler till textfiler. Uppgiften blir att skriva metoden

```
public static void doubleToText(String inFile,String outFile)
```

Den första parametern anger namnet på en binär fil som innehåller ett okänt antal tal av typen `double`, den andra parametern anger namnet på en textfil som skall skrivas av metoden. Talen i infilen skall läsas in till ett fält, därefter skall fältet sorteras, och sist skall talen i det sorterade fältet skrivas ut i textform, ett tal per rad, till utfilen. Sorteringen skall göras med standardmetoden `Arrays.sort`:

```
static void sort(double[] a)
```

Sorts the specified array of doubles into ascending numerical order.

Du kan anta att `FileUtilities` redan innehåller en metod för att mäta längden på en fil:

```
private static int length(String fileName)
```

Metoden returnerar längden, mätt i antalet bytes, hos filen med angivet filnamn. Konstanten `Double.SIZE` ger antalet bitar i den binära representationen av ett värde av typen `double`. Lösningen skall baseras på fält och sortering, andra strukturer som t.ex. map får ej användas. Ex. Så här kan ett stycke av en utfil se ut:

```
...
165.09518622909707
168.43766239639308
178.83306236636963
190.88998475424023
215.10695240071095
215.83468139012905
...
```

Plats för lösningen finns på nästa sida.

(12 p)

v.g.v.

```
public class FileUtilities {
    public static void doubleToText(String inFile,String outFile) {











    } ...
}
```

Uppgift 3

Ett bankkonto kan implementeras med en klass, `Account`, med följande metoder

`Account(x)`	konstruktor som initierar kontot med startsaldot x
`void put(x)`	sätter in x kronor på kontot
`void take(x)`	tar ut x kronor från kontot
`int getBalance()`	returnerar kontots saldo

a)
Implementera klassen `Account`. Vissa av metoderna bör vara synkroniserade, vilka? Om `take` anropas med ett uttagsbelopp som överskrider saldot, så skall metoden returnera först när tillräckligt belopp satts in på kontot (av ett annat objekt). Om `put` eller `take` anropas med ickepositivt argument skall undantaget `IllegalArgumentException` kastas. För att man skall kunna följa en simulering med kontotransaktioner skall dessutom `put` och `take` göra utskrifter som kolumnen till vänster visar exempel på:

...	
`3466 - 575 = 2891`	*saldo: 3466, uttag: 575, nytt saldo: 2891*
`2891 - 871 = 2020`	*uttag*
`2020 - 557 = 1463`	*uttag*
`1463 - 677 = 786`	*uttag*
`786 - 722 = 64`	*uttag*
...	*det saknas täckning för nästa uttag på 905 kronor*
`64 + 23554 = 23618`	*insättning*
`23618 - 905 = 22713`	*uttag*
`22713 - 596 = 22117`	*uttag*
...	

(12 p)

Med hjälp av kontoklassen, samt två trådklasser, kan vi nu konstruera ett enkelt simuleringsprogram. Scenariot är följande:Varje månad sätter en arbetsgivare in mellan 20000 och 30000 kronor i lön på kontot. Varje dag försöker kontohavaren ta ut mellan 1000 och 2000 kronor från kontot. Är pengarna slut får kontohavaren vänta tills det kommer in mer pengar. Vi låter en dag i verkligheten motsvaras av en sekund i simuleringen. Arbetsgivaren skall simuleras av klassen `Employer` och kontohavaren av klassen `Employed`. Följande kod sätter ihop de tre delarna och startar simuleringen:

```
Account account = new Account(30000);
Employer employer = new Employer(account);
Employed employed = new Employed(account);
employer.start();
employed.start();
```

Du kan välja att lösa antingen uppgift b eller c. Klasserna blir rätt lika och du får bara poäng för <u>en</u> av dem.

b)
Implementera klassen `Employer`. Klassen skall exekvera en tråd som var trettionde sekund sätter in ett slumpmässigt belopp enligt ovan i ett kontoobjekt.

(6 p)

c)
Konstruera klassen `Employed`. Klassen skall exekvera en tråd som varje sekund tar ut ett slumpmässigt belopp enligt ovan från ett kontoobjekt.

(6 p)

plats för lösningen →

```
class Random {
  …
   int nextInt(int n)
```

Returns a pseudorandom, uniformly distributed `int` value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

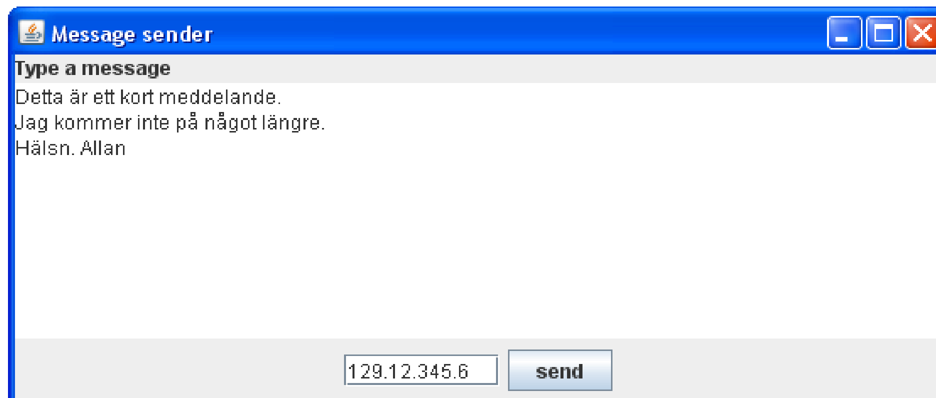
```
  …
}
```

```
public class Account {

}
```

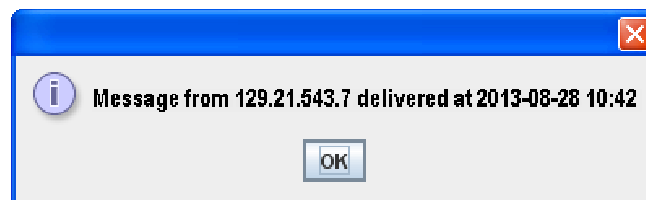
```
public class Employer (eller Employed)
{

}
```

Uppgift 4

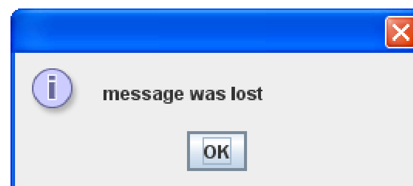
I denna uppgift skall du konstruera delar till sändarapplikationen i ett enkelt meddelandesystem där man får leveransbesked, ungefär på samma sätt som det kan fungera för SMS. [1]
Innehållet i en textarea skall sändas som datagram till en IP-adress, som anges i ett textfält, när användaren trycker på knappen **send**. Mottagarapplikationens portnummer antar vi är fixerat till 1837, så det behöver inte anges i GUI:t. Ex. Om vi antar att en sändarapplikation körs på en dator med IP-adressen 129.21.543.7, och en viss mottagare har IP-adress 129.12.345.6, kan sändning av ett meddelande och mottagning av leveransbesked se ut på följande sätt:



Vad mottagaren gör med meddelandet lämnar vi därhän, men mottagaren skall sända tillbaka ett kvittensdatagram med ett leveransbesked som innehåller *avsändarens* adress, samt tidpunkten då meddelandet togs emot. Avsändaren lyssnar efter leveransbesked på port 1838.
Leveransbeskedet skall visas i en pop-up-dialog hos avsändaren. (Texten i dialogen skall vara samma som innehållet i kvittensdatagrammet.)



Avsändaren väntar högst 10 sekunder på leveransbesked efter att meddelandet sänts. Om inget besked kommer inom denna tidsperiod skall meddelandet betraktas som förlorat och följande dialog visas



Tips: Placera mottagandet av leveransbesked i en privat metod så blir strukturen bättre.

(12 p)

plats för lösningen på nästa sida →

[1] För att undvika förvirring: Med "(av)sändaren" menas i denna uppgift den som sänder meddelanden. Med "mottagaren" menas den som tar emot meddelanden och sänder tillbaka leveransbesked.

```
public class Gui extends JFrame implements ActionListener
{
    private JTextArea textArea;
    private JTextField addressField;
    private JButton sendButton;
    private static int RECEIVER_PORT = 1837;
    private static int ACK_PORT = 1838;

    Gui() { makeFrame(); }

    private void makeFrame() {    Färdig metod, ingår ej i uppgiften.
        Sätter upp GUI:t. Adderar detta objekt som händelselyssnare på send-knappen.
    }
```

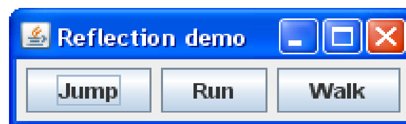
```
}
```
skriv på lösblad om inte lösningen fick plats här

Uppgift 5

Följande javagränssnitt är givet:

```
public interface Action {
    void execute();
}
```

En textfil innehåller namnen på ett okänt antal klasser som implementerar `Action`, ett namn per rad i filen. I samma katalog som namnfilen ligger motsvarande klassfiler. Om t.ex. ”`Jump`” finns i filen, så innehåller katalogen `Jump.class`, o.s.v.. Nedan finns en påbörjad klass som skall visa klassnamn på knappar i ett fönster. När man trycker på en knapp skall metoden `execute` anropas för en instans av klassen vars namn står på knappen.



I konstruktorn anropas metoden `makeButtons`. Uppgiften är att skriva färdigt metoden. `makeButtons` skall läsa filen, och för varje inläst klassnamn skapa en knapp med namnet på, samt ladda och instansiera klassen. Knappens lyssnarmetod skall anropa klassens `execute`-metod för den skapade instansen. Tips. Använd en `Scanner` för att läsa klassnamnen från namnfilen.

(6 p)

```
public class ReflectionDemo extends JFrame {
    public ReflectionDemo(String nameFile) {
        makeButtons(nameFile);
        pack(); setVisible(true);
    }
    private void makeButtons(String nameFile) {




    }
}
```