Programmeringsteknik för Bio1 och I1
Övning 6

Övningsgrupp 3 (Sal E33)

Johannes Hjorth
hjorth@nada.kth.se
Rum 4538 på plan 5 i D-huset
08 - 790 69 02

Kurshemsida:
http://www.nada.kth.se/kurser/kth/2D1310/

Övningsanteckningar och diagnostiska prov:
http://www.nada.kth.se/∼hjorth/teaching/

Dagens program

- Allmänna programmeringstips
  Hur skriver man bra kod?
- Kort om filhantering
  Hur läser vi information från fil?
- `StringTokenizer`
  Hur delar man upp en mening i enskilda ord?
- Snabb sortering
  Hur sorterar man objekt?
- Ett programmeringsexempel
  Hur gör man från början till slut?

Allmänna programmeringstips

Varför är det viktigt att först tänka igenom strukturen på programmet vi ska skriva?

- Skriva små program är lätt.
- Skriva stora program är svårt.

Genom att dela upp programmet i tillräckligt små oberoende bitar som kan skrivas var och en för sig blir det enklare att få det att fungera.

Varje bit kan testköras för sig och fel upptäcks fortare. Vi lägger ner lite mer tid på förarbete, men totalt sett går det snabbare.

Hur tillämpar vi det på J-uppgifterna?

Så här tillämpar vi det på J-uppgifterna

- Kopiera javakoden från er spec. till en java-fil
- Filen innehåller nu klasser, variabler och metodhuvuden.
- Se till att koden kompilerar. Metoder som har en returtyp måste returnera ett värde.

```
int sum(int a, int b) {
    return 1; // Vi vill bara få koden att kompilera
}
```

- När koden kompilerar, börja fylla i satser i en av metoderna. I vårt fall är det ganska enkelt.

```
int sum(int a, int b) {
    return a + b;
}
```

- Kompilera koden, testa att metoden fungerar som den ska med hjälp av en utskrift.

```
System.out.println(sum(2,3)); // Skriver ut 5
```

- Har det nu blivit fel så tittar vi direkt på den biten av kod vi ändrade senast.

Örjans budord

- Du skall aldrig skriva några satser innan du vet vad metoden som de står i är till för!
- Använd inte klassvariabler!
- Instansvariabler ska bara innehålla värden som har betydelse under hela instansens livslängd.
  Använd ej instansvariabler för att kommunicera mellan metoder.
- Namngivningen på synliga gränssnittet (metodnamn och parameterar) ska väljas utifrån anroparens perspektiv.
  En metod som beräknar en triangels area med Herons formel bör inte heta `heron` *utan* `triangelArea`.
- Använd metoder och klasser för att dela upp problemet så att man inte behöver veta hur andra metoder fungerar när man skriver "sin" metod.

Kort om filhantering

När programmet stängs glöms allt som ligger i minnet, det är därför bra att kunna spara vår information på ett mer permanent sätt. Då behöver vi kunna läsa och skriva till fil.

Vi kommer här gå igenom ett sätt för hur man läser in från fil med hjälp av `SimpleTextFileReader`, den intresserade rekommenderas att också titta på motsvarande metod för att skriva till fil.

Hur gör man då för att komma på hur den ska användas?

Har man inga exempel tittar man i källkoden!

`SimpleTextFileReader.java`

```
import java.io.*;

class SimpleTextFileReader {
    private BufferedReader in;

    // Skapa en förbindelse för att läsa från stdin.
    public SimpleTextFileReader() {
        in = new BufferedReader(new InputStreamReader(System.in));
    }

    // Skapa en förbindelse för att läsa från filen filename.
    public SimpleTextFileReader(String filename) {
        try {
            in = new BufferedReader(new FileReader(filename));
        } catch(FileNotFoundException e) {
            e.printStackTrace();
            System.exit(0);
        }
    }

    // Läser en textrad. Returnerar null vid filslut.
    public String readLine() {
        String str = null;
        try {
            str = in.readLine();
        } catch(IOException e) {
            e.printStackTrace();
            System.exit(0);
        }
        return str;
    }
```

`SimpleTextFileReader.java` **(forts)**

```
    // Stänger förbindelsen till filen.
    public void close() {
        try {
            in.close();
        } catch(IOException e) {
            e.printStackTrace();
            System.exit(0);
        }
    }
}
```

Observera att `SimpleTextFileReader` inte existerar i javas standardbibliotek, utan vi måste ladda ner den från kurshemsidan.

`http://www.nada.kth.se/kurser/kth/2D1311/03-04/j-uppgifter/hjaelpfiler/`

Den största fördelen med att använda klassen `SimpleTextFileReader` är att vi slipper ta hand om `IOExceptions`.

Hur anropar vi `SimpleTextFileReader`?

Vi skriver ett minimalistiskt program som läser in en textfil rad för rad och skriver ut den på skärmen.

```
import java.io.*;

class FilTest {

    public static void main(String[] args) {

        SimpleTextFileReader f = new SimpleTextFileReader("hej.txt");
        String s;

        s = f.readLine();          // läser in första raden

        while(s != null) {         // kollar om aktuell rad är tom
            System.out.println(s); // skriver ut raden
            s = f.readLine();      // läser in nästa rad
        }

        f.close();                 // stänger filen
    }
}
```

När vi kompilerar och kör koden, ser det ut så här.

```
>javac FilTest.java
>java FilTest
Hej detta är rad ett.
Detta är rad två.
Detta är rad tre, vår sista rad i filen.
```

Ett exempel

Vi vill nu skriva ett program som kan läsa in data om ett antal personer från fil och sedan skriva ut dem i åldersordning.

Vad behöver vårt program kunna göra?

- Läsa in personerna från fil
- Mellanlagra personerna i en `Vector`
- Sortera personerna efter ålder
- Skriva ut personerna i åldersordning

Vi vill skriva programmet stegvis så att det går att testköra varje del för sig. Det går då snabbare att finna de fel som uppkommer än om vi skriver hela programmet och sedan börjar felsöka.

Vi skapar en klass för en person

Eftersom vi ska arbeta med personer kan det vara vettigt att skapa en klass som har hand om all data om en enskild person.

I filen `Person.java` skriver vi,

```
class Person {

    private int födelseår;
    private String ögonfärg;
    private String kön;

    Person(int födelseÅr, String kön, String ögonfärg) {
        this.födelseår = födelseår;
        this.kön       = kön;
        this.ögonfärg  = ögonfärg;
    }

    public String toString() {
        return "En " + ögonfärg + "ögd " + kön + " född " + födelseår;
    }
}
```

Vi har infört tre instansvariabler samt skrivit en konstruktor och en `toString`-metod för utskrift.

Vi kommer successivt utvidga den här klassen medan vi kodar. Det här är bara för att komma igång.

Test av vår nya klass

För att kontrollera att vår `Person`-klass är korrekt skriver vi en enkel testklass i `TestPerson.java`

```
import java.io.*;

class TestPerson {

    public static void main(String[] inparametrar) {

        Person p = new Person(1978, "man", "blå");
        System.out.println(p);

    }
}
```

Om vi kompilerar och kör den här koden, så får vi följande utskrift

```
>javac TestPerson.java
>java TestPerson
En blåögd man född 0
```

Koden är tillräckligt korrekt för att kompilatorn ska godkänna den, men det är uppenbarligen något fel med mannens födelseår.

Kan ni se felet?

Felsökning!

Vi börjar med att kolla de enklaste möjligheterna till fel först så vi kan utesluta dem.

En snabb blick på `toString`-metoden och vi konstaterar att den skriver ut rätt variabel.

```
public String toString() {
    return "En " + ögonfärg + "ögd " + kön + " född " + födelseår;
}
```

Alltså måste instansvariabeln som innehåller året fått fel värde någonstans, men var?

I `main`-metoden skickar vi in 1978 till konstruktorn.

```
public static void main(String[] inparametrar) {

    Person p = new Person(1978, "man", "blå");
    System.out.println(p);

}
```

Jakten fortsätter...

Vi tar och tittar närmare på konstruktorn,

```
Person(int födelseÅr, String kön, String ögonfärg) {
    this.födelseår = födelseår;
    this.kön       = kön;
    this.ögonfärg  = ögonfärg;
}
```

Tilldelningsraden verkar se bra ut,

```
this.födelseår = födelseår;
```

Tittar vi på konstruktorns formellaparametrar ser vi att den första heter `födelseÅr` med stort Å.

```
Person(int födelseÅr, String kön, String ögonfärg)
```

Både `this.födelseår` och `födelseår` är samma variabel. Vi har alltså tilldelat instansvariabeln sitt eget värde och sedan kastat bort den lokala variabeln `födelseÅr`. Detta är felet!

Lösning: Vi ändrar det stora Å:et till ett litet.

Vi skriver ett skal

Vi skapar en mall för hur klassen `Personer` ska se ut. Metoderna får förklarande namn, vi vet dock ännu inte exakt hur de ska skrivas inuti.

```
import java.io.*;
import java.util.*;

class Personer {

    private Vector personer;     // här lagras alla personer

    Personer(String filnamn) {   // konstruktor
        personer = new Vector();
        läsInFrånFil(filnamn);
        sorteraÅlder();
    }

    private void läsInFrånFil(String filnamn) { }

    private void sorteraÅlder() { }

    public String toString() { return "text"; }

    public static void main(String[] args) {
        Personer p = new Personer("personer.txt");
    }
}
```

Notera att vi i main metoden instansierar ett objekt av klassen `Personer`. Detta är fullt tillåtet.

Inläsning från fil

Med `SimpleTextFileReader` kan vi enkelt läsa in flera personer från en fil.

I filen `personer.txt` står det,

```
Gustav 2002 man brun
Johan 1983 man blå
Birgit 1921 kvinna blå
Igor 1899 man röd
```

Eftersom varje rad innehåller flera fält med information kommer vi också få användning av `StringTokenizer` som delar upp en mening i ord.

Hur använder vi `StringTokenizer`?

StringTokenizer

Svaret till hur vi använder `StringTokenizer` finner vi i Java API.

`http://www.sgr.nada.kth.se/unix/docs/java/api/`

Constructor Summary

StringTokenizer(String str)
Constructs a string tokenizer for the specified string.

Method Summary

int countTokens()
Calculates the number of times that this tokenizer's nextToken method can be called before it generates an exception.

boolean hasMoreTokens()
Tests if there are more tokens available from this tokenizer's string.

String nextToken()
Returns the next token from this string tokenizer.

Vi skriver `läsaInFrånFil`-metoden

Observera alla utskrifter, de hjälper oss felsöka ifall något skulle bli fel.

```
private void läsInFrånFil(String filnamn) {
    SimpleTextFileReader f = new SimpleTextFileReader(filnamn);
    StringTokenizer st;
    Person p;

    String namn, kön, ögonfärg, s;         // temporära variabler
    int ålder;

    s = f.readLine();                      // läser in första raden

    while(s != null) {                     // kollar om aktuell rad är tom
        System.out.println("Inläst rad: " + s);

        st       = new StringTokenizer(s); // delar upp strängen

        namn     = st.nextToken();
        ålder    = Integer.parseInt(st.nextToken());
        kön      = st.nextToken();
        ögonfärg = st.nextToken();         // hämtar sista delsträngen

        p = new Person(namn, ålder, kön, ögonfärg);

        System.out.println("Persondata: " + p);

        personer.addElement(p);            // spara personen i vektorn

        s = f.readLine();                  // läser in nästa rad
    }

    f.close();                             // stänger filen
}
```

Kompilera och testkör

Vi kompilerar och testkör för att se att vår inläsning fungerar som den ska.

```
>javac Personer.java
>java Personer
Inläst rad: Gustav 2002 man brun
Persondata: Gustav är en brunögd man född 2002
Inläst rad: Johan 1983 man blå
Persondata: Johan är en blåögd man född 1983
Inläst rad: Birgit 1921 kvinna blå
Persondata: Birgit är en blåögd kvinna född 1921
Inläst rad: Igor 1899 man röd
Persondata: Igor är en rödögd man född 1899
```

Det ser ut som om inläsningen fungerar, då är nästa steg att kolla att vektorn `personer` har fyllts på korrekt och sedan att sortera den!

Personers `toString`-metod

Uppgiften för `toString`-metoden är att iterera igenom hela vektorn `personer` och lägga till var och en av dem till strängen som sedan returneras.

En `String` kan inte ändras efter att den skapats. Därför använder vi oss av `StringBuffer`-klassen och konverterar resultatet till `String` i slutet.

```
public String toString() {

    StringBuffer s = new StringBuffer();
    Person p;

    for(int i = 0; i < personer.size(); i++) {
        p = (Person) personer.elementAt(i);
        s.append(i + ": " + p.toString() + "\n");
    }

    return s.toString(); // Vi använder StringBuffers toString metod
}
```

Vi lägger också till en utskrift av `p` i `main`-metoden.

```
public static void main(String[] args) {
    Personer p = new Personer("personer.txt");
    System.out.println(p);
}
```

Vi kompilerar och testkör

Vi kompilerar och testkör vårt program,

```
>javac Personer.java
>java Personer
Inläst rad: Gustav 2002 man brun
Persondata: Gustav är en brunögd man född 2002
Inläst rad: Johan 1983 man blå
Persondata: Johan är en blåögd man född 1983
Inläst rad: Birgit 1921 kvinna blå
Persondata: Birgit är en blåögd kvinna född 1921
Inläst rad: Igor 1899 man röd
Persondata: Igor är en rödögd man född 1899
0: Gustav är en brunögd man född 2002
1: Johan är en blåögd man född 1983
2: Birgit är en blåögd kvinna född 1921
3: Igor är en rödögd man född 1899
```

Det ser bra ut, vektorn med personer skrivs ut på rätt sätt. Tjusigt!

Nu är det bara sorteringen som återstår...

Snabb sortering

Java har inbyggda rutiner för sortering. Det enda vi behöver göra är att berätta för algoritmen hur man jämför två personer, det vill säga vilken som ska komma först av de två.

Detta gör vi enklast genom att modifiera `Person`-klassen. Orden `implements Comparable` efter klassnamnet på nästa sida betyder att vi lovar att implementera metoden `compareTo` vilken behövs för att sorteringen ska fungera.

Två nya metoder har tillkommit i `Person`, först

```
public int getFödelseår() {
    return födelseår;
}
```

och sedan `compareTo`-metoden som vi lovade implementera

```
public int compareTo(Object o) {
    Person p = (person) o;

    return födelseår - p.getFödelseår();
}
```

Sortera!

Eftersom vi nu har en naturlig ordning på våra `Person` objekt kan vi använda de inbyggda sorteringsmetoderna för att ordna dem i rätt ordning.

Själva sorteringen utförs genom anropet

```
Collections.sort(personer);
```

vilket vi gör i `sorteraÅlder`-metoden i `Personer`-klassen.

Hur ser de färdiga klasserna ut?

Vår sorterbara `Person`-klass

```
class Person implements Comparable {

    private String namn;
    private int födelseår;
    private String ögonfärg;
    private String kön;

    Person(String namn, int födelseår, String kön, String ögonfärg) {
        this.namn      = namn;
        this.födelseår = födelseår;
        this.kön       = kön;
        this.ögonfärg  = ögonfärg;
    }

    public String toString() {
        return namn + " är en " + ögonfärg + "ögd " + kön + " född " + födelseår;
    }

    public int getFödelseår() {
        return födelseår;
    }

    public int compareTo(Object o) {
        Person p = (Person) o;

        return födelseår - p.getFödelseår();
    }
}
```

Den uppdaterade `Personer`-klassen

```
import java.io.*;
import java.util.*;

class Personer {

    private Vector personer;

    Personer(String filnamn) {
        personer = new Vector();
        läsInFrånFil(filnamn);
        sorteraÅlder();
    }

    private void läsInFrånFil(String filnamn) {
        SimpleTextFileReader f = new SimpleTextFileReader(filnamn);
        StringTokenizer st;
        String namn, kön, ögonfärg, s;
        int ålder;

        s = f.readLine();          // läser in första raden

        while(s != null) {         // kollar om aktuell rad är tom
            st       = new StringTokenizer(s);

            namn     = st.nextToken();
            ålder    = Integer.parseInt(st.nextToken());
            kön      = st.nextToken();
            ögonfärg = st.nextToken();

            personer.addElement(new Person(namn, ålder, kön, ögonfärg));

            s = f.readLine();      // läser in nästa rad
        }

        f.close();                 // stänger filen
    }

    private void sorteraÅlder() {
        Collections.sort(personer);
    }

    public String toString() {

        StringBuffer s = new StringBuffer();
        Person p;

        for(int i = 0; i < personer.size(); i++) {
            p = (Person) personer.elementAt(i);
            s.append(i + ": " + p.toString() + "\n");
        }

        return s.toString();
    }

    public static void main(String[] args) {
        Personer p = new Personer("personer.txt");
        System.out.println(p);
    }
}
```

Här har nu metoden `sorteraÅlder` blivit implementerad.

Vi har även tagit bort en del av utskrifterna till skärmen från `läsInFrånFil` eftersom vi vet att det fungerar.

Vi kompilerar och kör

Vi kompilerar vårt program och kör...

```
>javac Personer.java
>java Personer
0: Igor är en rödögd man född 1899
1: Birgit är en blåögd kvinna född 1921
2: Johan är en blåögd man född 1983
3: Gustav är en brunögd man född 2002
```

Det fungerar! Personerna är sorterade i rätt ordning.

Slutet gott allting gott. :-)

Kom ihåg!

- Tänk igenom hur programmet ska fungera först
- Bygg upp programmet stegvis
- Kompilera om så fort du ändrat något
- Rätta nya fel direkt