Institutionen för
Data- och informationsteknik
JSk

TENTAMEN
2010-01-11

OBJEKTORIENTERAD PROGRAMMERING
för Z1 (TDA540)

OBS! Det kan finnas kurser med samma eller liknande namn på olika utbildningslinjer. Denna tentamen gäller *endast* för den eller de utbildningslinjer som anges ovan. Kontrollera därför noga att denna tentamen gäller för den utbildningslinje du själv går på.

TID 14.00 - 18.00

Ansvarig: Jan Skansholm, tel. 772 10 12 eller 0707-163230

Betygsgränser: Sammanlagt maximalt 60 poäng.
På tentamen ges graderade betyg:
3:a 24 poäng, 4:a 36 poäng och 5:a 48 poäng

Hjälpmedel: Skansholm, *Java direkt med Swing*, valfri upplaga, Studentlitteratur.
(Understrykningar och mindre anteckningar i boken är tillåtna.)

Inga kalkylatorer är tillåtna.

Tänk på:

- att skriva tydligt och disponera papperet på ett lämpligt sätt.
- att börja varje ny (del)uppgift på nytt blad. Skriv endast på en sida av papperet
- Skriv den (anonyma) kod du fått av tentamensvakten på *alla* blad.

De råd och anvisningar som givits under kursen skall följas vid programkonstruktionerna. Det innebär bl.a. att onödigt komplicerade, långa och/eller ostrukturerade lösningar i värsta fall ej bedöms.

Uppgift 1) a) Vilket påstående är korrekt om `MemberList` på basis av vad som visas nedan? För poäng krävs att du motiverar ditt svar!

```
import java.util.*;
public class MemberList {
  private LinkedList<Person> list;
  private String name;
  public MemberList(String name) { this.name = name; }
  public void add(Person p) { list.add(p); }
}
```

a. Klassen går inte att kompilera.
b. Det går inte att skapa objekt av klassen.
c. Exekveringen riskerar att avbrytas om objekt av klassen används.
d. Klassen är perfekt.

(2 p)

b) Klassen `Int` ser ut på följande sätt:

```
public class Int {
  private int x;
  public Int() { x = 0; }
  public void set( int x ) { this.x = x; }
  public int get() { return x; }
}
```

Vilken utskrift ger kodavsnittet nedan?

```
ArrayList<Int> list = new ArrayList<Int>();
Int n = new Int();
for ( int j = 0; j < 3; j++ )
  list.add( n );
int k = 1;
for ( Int x : list )
  x.set( k++ );
for ( Int x : list )
  System.out.println( x.get() );
```

(3 p)

c) Som du vet får en instansmetod anropa andra metoder (både instansmetoder och klassmetoder) i samma klass direkt, utan att man skriver något objektnamn framför namnet på metoden som anropas. Men varför får en klassmetod inte anropa en instansmetod i samma klass på detta enkla sätt?

(2 p)

d) Standardklassen `Point` har som bekant de publika instansvariablerna `x` och `y`. Om vi skriver

```
int[] a = new int[5];
a[3] = 7;
```

fungerar det precis som vi tänkt. Men om vi skriver

```
Point[] b = new Point[5];
b[3].x = 7;
```

får vi vid körning en felsignal och körningen avbryts vid sista raden. Förklara varför! Hur rättar man?

(2 p)

e) Varför kan man inte ange att en viss `LayoutManager` skall användas för en komponent av klassen `Button`?

(1 p)

Uppgift 2) En ekvation av formen Ax + By + Cz = D, där A, B, C och D kallas en diofantisk ekvation. (Diofantos var en grekisk matematiker som levde omkring 250 f. Kr.) Ekvationen kan ha noll eller flera lösningar. Vi är i denna uppgift bara intresserade av ekvationer i vilka A, B, C och D är heltal som är större än noll och vi söker bara sådana lösningar där x, y och z alla är större än eller lika med noll. Därmed vet vi t.ex. att x <= D/A. Skriv ett program som läser in de fyra talen A, B, C och D och sedan skriver ut alla lösningar enligt ovan och slutligen även en uppgift om antalet lösningar. Så här kan två körningar av programmet se ut:

```
Ange talen A, B, C och D: 2 4 6 7
0 lösningar!

Ange talen A, B, C och D: 2 4 6 8
0 2 0
1 0 1
2 1 0
4 0 0
4 lösningar!
```

(10 poäng)

Uppgift 3) Gör ett program som fungerar som en enkel form av en spelmaskin av typen "enarmad-bandit". Användargränssnittet skall vara som i följande figur.

Fönstret skall ha titeln *En bandit*. Överst skall finnas en knapp med texten **Spela**. Därunder en rad med tre utrymmen, som under spelets gång kommer att innehålla siffror mellan 0 och 9, men som från början är blanka. Underst finns en text som innehåller information om hur en spelomgång gick. När man klickar på knappen **Spela** genomförs en spelomgång, som består i att det slumpmässigt skapas tre heltal mellan 0 och 9 vilka placeras på mellanraden. Om de tre talen är lika visas texten **Vinst 50 kr**, annars texten **Ingen vinst**. Bilderna visar i tur och ordning: utgångsläget, en normal spelomgång och en omgång med vinst. Försök att få användargränssnittet att se ut ungefär som på bilderna. (Fontstorleken är 36 punkter.)

(12 poäng)

Uppgift 4) Du har säkert kommit i kontakt med standardprogram som komprimerar filer så att de inte tar så stor plats eller går fortare att överföra via nätet. I denna uppgift skall du skriva ett program som komprimerar textfiler på ett mycket enkelt sätt. Programmet skall läsa en textfil och skapa en annan fil som innehåller samma information som den ursprungliga filen, men i komprimerad form. Den nya filen skall ha samma namn som den ursprungliga filen, men med tillägget ".szip". (Står för 'simple zip'.) Namnet på filen som skall komprimeras skall läsas från kommandoraden.

I en vanlig textfil lagras som du vet varje tecken som en 8-bitars teckenkod, en s.k. *byte*. Vid komprimeringen skall varje sekvens av fler än 3 likadana tecken översättas till en kompaktare form. Om den ursprungliga filen t.ex. innehåller sekvensen `aaaaaa` så skall denna i den komprimerade filen översättas till följande tre bytes: `¥•a`. Byte nr 1 innehåller teckenkoden för yen-tecknet (unicode \u00A5). Denna byte används som en markör för att signalera att de två bytes som följer skall tolkas på ett speciellt sätt. (Naturligtvis blir det problem om den ursprungliga filen skulle råka innehålla ett yen-tecken, men det kan du bortse från här.) Byte nr 3 innehåller teckenkoden för det tecken som ingick i sekvensen (i det här exemplet tecknet `a`). Byte nr 2, den som ovan markerats som •, innehåller sekvensens längd, i detta exempel värdet 6 eftersom sekvensen innehöll 6 tecken. Byte nr 2 innehåller *inte* någon teckenkod, utan den innehåller helt enkelt ett binärt värde i intervallet 0 till 255. En sekvens kan alltså bestå av högst 255 tecken. (Den fil som generas av programmet kan alltså inte läsas av en vanligt texteditor eftersom en sådan skulle försöka tolka det som finns i de bytes som markerats med • som vanliga tecken. För att kunna läsa filen måste man skriva ett annat program som översätter filen tillbaka till vanlig form, men det ingår inte i uppgiften.)

Här kommer ett exempel. Om den ursprungliga filen innehåller

```
Thisssssssss is aaaaaa
testtt 2222255555555555
```

så skall den komprimerade filen innehålla

```
Thi¥•s is ¥•a
testtt ¥•2¥•5
```

där de bytes som markerats med • innehåller värdena 8, 6, 5 resp. 11. Du ser att om en sekvens består av färre tecken än 4 så lönar det sig inte att komprimera den.

Använd standardklasserna `BufferedReader` och `BufferedWriter`. *Tips*: I klassen `BufferedWriter` finns metoden `write`. Denna skriver ut en byte. Metoden får som parameter en **`int`**, vilken kommer att skrivas till utströmmen. (I normala fall innehåller parametern en teckenkod.) På motsvarande sätt finns det i klassen `BufferedReader` en metod med namnet `read` som läser en byte i taget från inströmmen och som returnerar det värde som finns i denna byte. Returtypen är **`int`**.

(12 poäng)

Uppgift 5) Standardklassen `String` har som du vet egenskapen att det inte går att ändra ett visst `String`-objekt. Varje gång man vill lägga en text i ett `String`-objekt skapas det ju et *nytt* objekt. Detta kan förstås vara lite ineffektivt om man arbetar med texter som skall ändras ofta. I Java finns emellertid även klasserna `StringBuilder` och `StringBuffer`. Objekt av dessa klasser innehåller också texter men texterna är ändringsbara. I denna uppgift skall du skriva en egen (kraftigt förkortad) version av klassen `StringBuilder`. Din klass skall heta `MyStringBuilder`.

I din klass skall texten lagras i ett internt fält (array) med komponenter av typen **`char`**. Antalet komponenter i fältet kallas *kapaciteten*. I fältet kan man lagra texter vilkas längd inte överskrider kapaciteten. Om man försöker lagra en längre text skall kapaciteten automatiskt ökas (vi återkommer till detta).

I klassen `MyStringBuilder` skall det finnas två konstruktorer: Den första skall vara parameterlös och när den används skall den initiera det nya objektet så att kapaciteten blir lika med 16. Den andra konstruktorn skall ha en parameter av typen `String`. När denna konstruktor används skall kapaciteten bli lika med parameterns längd plus 16. Dessutom skall den text som finns i parametern placeras i början av det interna fältet.

Förutom konstruktorerna skall klassen `MyStringBuilder` ha följande metoder:

- `length()`, ger längden av den text som finns i objektet,
- `capacity()`, ger objektets kapacitet,
- `ensureCapacity(k)`, utökar (om så behövs) objektets kapacitet så att den blir större än eller lika med `k`. Kapaciteten skall (ev. upprepade gånger) fördubblas tills villkoret är uppfyllt.
- `insert(p,s)`, skjuter in texten `s` (en `String`) med början i position `p` (en **`int`**), utökar kapaciteten om så behövs. Skall kontrollera att `p` är större än eller lika med noll och mindre än eller lika med `length()`. Om så inte är fallet skall en exception av standardklassen `IndexOutOfBounds` genereras. Om `p` är mindre än `length()` måste en del av den tidigare texten förskjutas för att bereda plats för `s`.
- `substring(p,m)`, ger som resultat en `String` som består av tecknen i positionerna `p` till `m-1`. Om `p` eller `m` är negativa eller större än `length()` eller om `p > m` så skall en exception av standardklassen `IndexOutOfBounds` genereras.
- `append(s)`, lägger till texten `s` (en `String`) sist, utökar kapaciteten om så behövs.
- `toString`, returnerar texten i objektet som en `String`.

Tips: Det två sista metoderna kan göras väldigt enkla om du är lite smart och utnyttjar annat du gjort.

(16 p)