Institutionen för
Data- och informationsteknik

TENTAMEN

OBJEKTORIENTERAD PROGRAMVARUUTVECKLING

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 08.30-12.30

Ansvarig: Jan Skansholm

Förfrågningar: Jan Skansholm

Betygsgränser: Sammanlagt maximalt 60 poäng.
På tentamen ges graderade betyg:.
G 24 poäng, VG 48 poäng

Hjälpmedel: Skansholm, *Java direkt med Swing*, valfri upplaga, Studentlitteratur

Inga kalkylatorer är tillåtna.

Tänk på:

- att skriva tydligt och disponera papperet på ett lämpligt sätt.
- att börja varje ny 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) Du kan anta att klassen `Person` finns, är felfri och är synlig.
Vilket påstående är då korrekt om `C` på basis av vad som visas nedan?
För poäng krävs att du motiverar ditt svar!

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

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

(2 p)

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

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

Vilken utskrift ger kodavsnittet nedan?

```
List<Value> list = new LinkedList<>();
Value i = new Value();
for ( int j = 0; j < 3; j++ )
  list.add( i );
int k = 3;
for ( Value x : list )
  x.set( k-- );
for ( Value x : list )
  System.out.println( x.get() );
```

(2 p)

c) I ett program har man lagt följande rader:

```
String sa = "ABC";
String sb = sa;
String sc = "abc".toUpperCase();
if (sa==sc)
  System.out.println("lika");
else
  System.out.println("olika");
if (sa==sb)
  System.out.println("lika");
else
  System.out.println("olika");
```

Ange vilken utskrift programmet bör ge och *förklara varför*!

(2 p)

Uppgift 2) Skriv ett program `Find` som läser en befintlig textfil och söker efter de rader i filen som innehåller en viss text. De funna raderna skall kopieras till en ny fil. Indata till programmet skall ges som argument på kommandoraden. (Programmet skall alltså *inte* läsa indata från något kommandofönster eller via `System.in`.) När man startar programmet skall man på kommandoraden ge följande tre argument: namnet på den befintliga filen, namnet på den nya filen samt den text man söker. Man kan t.ex. ge kommandot

```
java Find bilar.txt volvo.txt Volvo
```

Här kommer alla de rader i filen `bilar.txt` vilka innehåller texten `Volvo` att kopieras till filen `volvo.txt`. Programmet *skall* kontrollera att antalet argument är korrekt *och* att filerna går att öppna. Om något skulle vara fel skall en felutskrift ges och programmet avslutas.

(10 p)

Uppgift 3) a) Skriv en klass `Verktyg` som innehåller en klassmetod med namnet `platsFör`. Metoden `platsFör` skall ha två heltalsparametrar `k` och `n`. Du får förutsätta att parametern `k` innehåller ett positivt ensiffrigt värde, dvs. ett värde i intervallet 0 till 9, och att `n` >= 0. Metoden skall undersöka om, och i så fall var, den siffra som finns i `k` förekommer i talet `n`. Som resultat skall placeringen för siffran `k` anges. Placeringen räknas från höger i talet. Numreringen börjar på 0. Om en siffra förekommer flera gånger skall den första förekomsten (från höger räknat) anges. Om siffran `k` inte finns i talet `n` skall resultatet -1 ges. Det är i denna uppgift *inte* tillåtet att göra om heltalen till typen `String`.

Några exempel: k=6 och n=2356 skall ge resultatet 0, k=6 och n=6350 resultatet 3, k=6 och n=6360 resultatet 1 samt k=6 och n=4350 resultatet -1.

(5 p)

b) Utöka klassen `Verktyg` med en klassmetod med namnet `innehåller`. Denna metod skall ha samma parametrar som metoden `platsFör`. Samma förutsättningar för parametrarna gäller också. Metoden `innehåller` skall *förutom* att beräkna placeringen för siffran `k` i `n` även undersöka hur många gånger siffran `k` förekommer i `n`. Den skall alltså beräkna två resultat. Du får själv hitta en lösning för hur detta skall gå till. Det är inte heller i denna uppgift tillåtet att göra om heltalen till typen `String`

Några exempel: Om k=6 och n=2356 skall resultaten 0 och 1 ges. Om k=6 och n=6350 skall resultaten 3 och 1 ges. Om k=6 och n=6360 skall resultaten 1 och 2 ges. Om k=6 och n=4350 skall resultaten -1 och 0 ges.

(5 p)

Uppgift 4) Uppgiften är att konstruera en klass `ProduktId` vars uppgift är att hålla reda på unika id-nummer för olika produkter. Varje objekt av denna klass skall innehålla ett namn (en `String`) och ett id-nummer (en **`int`**). Id-numren skall vara unika så att inte två eller flera produkter har samma nummer. Om man skriver siffror (slarvigt) för hand kan lätt siffrorna 4 och 9 förväxlas. Samma sak gäller för siffrorna 1 och 7. Av denna anledning har man bestämt att produkternas id-nummer inte skall få innehålla siffrorna 4 och 7.

Klassen skall innehålla instansmetoderna `avläsNamn` och `avläsId` som ger en viss produkts namn resp. id-nummer. Dessutom skall det finnas en klassmetod `avläsAntal` som ger det totala antalet objekt av typen `ProduktId`.

Det måste naturligtvis finnas en konstruktor för klassen `ProduktId`. Denna skall som enda parameter få en `String` med produktens namn. Konstruktorn skall hålla reda på hur många objekt av typen `ProduktId` som skapas. Konstruktorn skall dessutom själv tilldela ett nytt, unikt id-nummer till varje nytt objekt. För att få fram ett nytt id-nummer kan man prova med att addera 1 till det id-nummer man gav det senaste `ProduktId`-objektet. Det nya id-numret får emellertid inte innehålla siffrorna 4 och 7. För att prova om ett nummer innehåller en viss siffra får du gärna använda metoden `platsFör` i klassen `Verktyg` från uppgift 3a. Om man provar ett nummer men finner att det innehåller en otillåten siffra är det i de flesta fall ineffektivt att bara öka numret med 1 och göra ett nytt prov. Antag t.ex. det förra numret man tilldelade ett `ProduktId`-objekt var 33999. Man ökar då numret med 1 och finner att 34000 är otillåtet. Det är förstås ineffektivt att fortsätta att prova med 34001 osv. Istället skall man addera 1000 och prova med 35000 som visar sig vara tillåtet. Gör på det sättet (inte bara för tusental utan för alla potenser av tio).

(12 p)

Uppgift 5) Utgå från följande två klasser `Klockslag` och `Observation`:

```
public class Klockslag implements Comparable<Klockslag> {
  int tim;
  int min;

  public int compareTo(Klockslag k) {
    if  (tim < k.tim || (tim == k.tim && min < k.min))
      return -1;
    else if (tim > k.tim || (tim == k.tim && min > k.min))
      return 1;
    else
      return 0;
  }

  @Override
  public String toString() {
    return String.format("%02d:%02d", tim, min);
  }
}

public class Observation implements Comparable<Observation> {
  int nr;
  Klockslag tid = new Klockslag();
  double temp;
  double vindhast;
  int vindrikt;

  public int compareTo(Observation annan) {
    return tid.compareTo(annan.tid);
  }
}
```

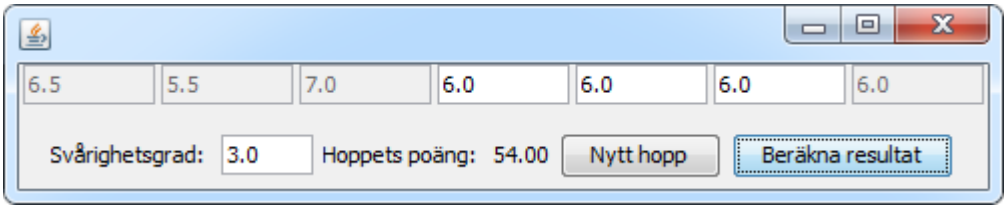
Antag sedan att du har en textfil `obs.txt` som innehåller väderobservationer. Varje rad i filen innehåller information om en observation och har formen:

```
stationsnr tim min temperatur vindhast vindriktning
```

Skriv ett program som läser informationen i filen och lagrar den i en lista med element av typen `Observation`. Låt programmet sortera listan så att observationerna hamnar i tidsordning. Skriv sedan ut väderstationens nummer, tidpunkten och temperaturen för alla observationer. När detta gjorts skall listan sorteras om så att den istället i första hand sorteras efter väderstationens nummer. *Tips*: Använd en extern jämförare. Programmet skall slutligen åter skriva ut stationsnummer, tidpunkt och temperatur för alla observationer.

(10 p)

Uppgift 6) Vid internationella simhoppstävlingar bedöms varje hopp av sju domare som ger poäng på en skala från 0 till 10. Man kan bara ge hela och halva poäng, t.ex. 6.0 och 6.5. Beräkning av hoppets poäng görs på följande sätt: Man bortser från de två högsta och de två lägsta domarpoängen. Därefter summeras de återstående tre domarpoängen. Hoppets poäng får man fram genom att multiplicera den erhållna summan med ett tal som anger hoppets svårighetsgrad (i normala fall ett tal med en decimal i intervallet 1.0 till 5.0). Uppgiften är att skriva ett program som läser in domarpoängen och svårighetsgraden och som visar hoppets poäng. Programmet skall ha ett grafiskt användargränssnitt som ser ut ungefär som i följande figur.



Längst upp finns sju rutor i vilka man skall fylla i domarpoängen. Svårighetsgraden skriver man in i rutan *Svårighetsgrad*. När man vill beräkna poängen för ett hopp klickar man först på knappen *Nytt hopp*. Då skall alla rutorna för domarpoäng och rutan för svårighetsgraden bli blanka. Talet efter texten *Hoppets poäng* skall också tas bort. När man sedan har fyllt i alla rutorna klickar man på knappen *Beräkna resultat*. Då skall programmet räkna ut hoppets poäng och visa denna. Programmet skall då dessutom markera vilka domarpoäng som var högst och lägst och som därför räknades bort. Det kan göras genom att motsvarande rutor inaktiveras som i exemplet ovan. (Dessa rutor skall naturligtvis aktiveras igen när man skall läsa in indata för nästa hopp.)

För enkelhets skull får du förutsätta att alla indata matas in på ett korrekt sätt.

(12 p)