TENTAMEN OOP 2014-01-19

ANVISNINGAR

- Påbörja varje ny uppgift på nytt blad.
- Skriv endast på ena sidan av bladen.
- Skriv tydligt - oläsbara svar beaktas ej.

BETYGSÄTTNING

Max antal poäng är 30.

För att bli godkänd på tentan (minst betyg E) krävs dels minst 4 poäng sammanlagt på uppgift 1 och uppgift 2 och dels minst 15 poäng sammanlagt på hela tentan.

För högre betyg krävs:

- Betyg D: minst 18 poäng samt högst en uppgift med 0 poäng.
- Betyg C: minst 21 poäng samt ingen uppgift med 0 poäng.
- Betyg B: minst 24 poäng samt ingen uppgift med 0 poäng.
- Betyg A: minst 27 poäng samt uppgifterna lösta med korrekt användande av objektorienterade principer (t.ex. inkapsling, ej upprepning av kod).

Betyget Fx med möjlighet att komplettera ges till studenter som fått 12-14 poäng eller som fått mer än 14 poäng men missat att få 4 poäng på uppgift 1+2.

HJÄLPMEDEL

De enda tillåtna hjälpmedlen är en (1) valfri bok om Java och en (1) bok som inte behandlar programmering i någon form.

Lycka till!

Lösningsförslag läggs upp i Moodle senast tre arbetsdagar efter tentatillfället.

UPPGIFT 1: KODFÖRSTÅELSE IMPERATIV PROGRAMMERING (6 POÄNG)

Om man exekverar följande programrader, vad kommer att skrivas ut på skärmen? Du skall i dina svar vara noga med vad som skrivs på vilken rad, alltså beakta skillnaden mellan print och println.

DELUPPGIFT A (2 POÄNG)

```
int tal = 4;
switch (++tal) {
case 4:
   System.out.println(tal);
   break;
case 5:
   System.out.println(tal++);
default:
   System.out.println(++tal);
}
int tal2 = tal + 3;
System.out.println(tal + tal2);
```

KORREKT SVAR

```
5
7
17
```

VARFÖR BLIR SVARET VAD DET BLIR?

Den första raden sätter variabeln tal = 4. ++tal på rad två ökar tal till 5 **innan** valet av case sker. I fall 5 skriver vi ut tal++, alltså 5, som **efter** utskriften ökas till 6. Eftersom det inte finns något break så faller vi igenom till default och skriver ut ++tal, alltså 7 eftersom tal ökas **innan** utskriften.

Efter switch-satsen sätts tal2 till 7 + 3, alltså 10, och vi skriver ut 7+10, alltså 17. Alla utskrifter använder println, så de kommer på varsin rad.

DELUPPGIFT B (2 POÄNG)

```
int[] arr = new int[5];

for(int i=1; i<=arr.length; i+=2){
   arr[arr.length-i]=i;
}

for(int x: arr){
   System.out.print(x);
}
```

KORREKT SVAR

50301

VARFÖR BLIR SVARET VAD DET BLIR?

Den första raden skapar en array med fem platser:

0	0	0	0	0
0	1	2	3	4

När man skapar en array av heltal så sätts samtliga värden till noll från början.

Den första loopen går tre varv och variabeln i sätts i tur och ordning till 1, 3 och 5, Under det första varvet så sätts plats arr.length-1 (4) till 1, under det andra arr.length-3 (2) till 3 och slutligen arr.length-5 (0) till 5 vilket ger oss följande array som sedan skrivs ut av den andra loopen:

5	0	3	0	1
0	1	2	3	4

DELUPPGIFT C (2 POÄNG)

```
for (int i = 0; i < 5; i++) {

   int j = 5 - i;

   while (j > 0) {
      if (j % 2 == 0) {
         System.out.print(j);
      }
      j--;
   }

   System.out.println();
}
```

KORREKT SVAR

```
42
42
2
2
TOM RAD
```

VARFÖR BLIR SVARET VAD DET BLIR?

Den här koden går att följa för hand, men blir mycket enklare om man funderar på vad koden inuti for-loopen gör. While-loopen skriver ut alla jämna tal från j och ner till 2. I det första varvet på forloopen blir j 5-0, alltså 5, och vi skriver därför ut först 4 och sen 2. I det andra varvet blir j 5-1,

alltså 4, och vi gör samma sak. I det tredje varvet blir j 5-2, alltså 3, och vi skriver bara ut 2, vilket även gäller det fjärde varvet när j är 2. I det sista varvet i forloopen blir j 5-4, alltså 1, och ingen siffra skrivs ut, men däremot radbrytningen.

UPPGIFT 2: KODFÖRSTÅELSE KLASSER OCH OBJEKT (6 POÄNG)

Givet nedanstående tre klasser, vad kommer att skrivas ut om man kör programmet?

```
class SuperClass {
   private String data;

   public SuperClass(int data) {
      this.data = "" + data + data;
   }

   public String toString() {
      return "" + data;
   }
}
```

```
class SubClass extends SuperClass {
   private int data;
   private static int merData = 79;

   public SubClass(int data) {
      super(data + data);
      this.data = data;
      merData = data + data + data;
   }

   public String toString() {
      return super.toString() + merData;
   }
}
```

```
class MainClass {
   public static void main(String[] args) {
      SuperClass c1 = new SuperClass(3);
      SuperClass c2 = new SubClass(4);
      SubClass c3 = new SubClass(5);

      System.out.println(c1);
      System.out.println(c2);
      System.out.println(c3);
   }
}
```

KORREKT SVAR

```
33
8815
101015
```

VARFÖR BLIR SVARET VAD DET BLIR?

I main-metoden skapas tre stycken objekt, en instans av SuperClass och två av SubClass. Låt dig inte luras av att typen på c2 är SuperClass, det är bara typen på objektet som spelar roll för beteendet. (Variabelns typ spelar däremot roll för vilket gränssnitt man kan använda sig av för interaktion med objektet, men det är inte relevant i denna uppgift.)

När c1 skapas så får den en instansvariabel data som sätts till 33 eftersom + i konstruktorn i superklassen avser strängkonkatenering, inte addition.

När c2 skapas så blir det lite svårare. Här finns det förutom parametrarna två instansvariabler som bägge heter data, en sträng i superklassen och en int i subklassen. + betyder också olika saker vid olika tillfällen. Låt oss följa med 4:an på dess väg:

1. Parametern data i subklassens konstruktor sätts till 4
2. Superklassens konstruktor anropas med 4+4, alltså 8 som argument
3. Parmetern data i superklassens konstruktor sätts till 8
4. Instansvariabeln data i superklassen sätts till ””+8+8 = strängen 88
5. Superklassens konstruktor är klar och vi återgår till subklassens
6. Subklassens instansvariabel data sätts till 4, värdet på parametern data
7. Den statiska variabeln merData sätts till 4+4+4=12

När c3 skapas sker samma sak:

1. Parametern data i subklassens konstruktor sätts till 5
2. Superklassens konstruktor anropas med 5+5, alltså 10 som argument
3. Parmetern data i superklassens konstruktor sätts till 10
4. Instansvariabeln data i superklassen sätts till ””+10+10 = strängen 1010
5. Superklassens konstruktor är klar och vi återgår till subklassens
6. Subklassens instansvariabel data sätts till 5, värdet på parametern data
7. Den statiska variabeln merData sätts till 5+5+5=15

Därefter anropas toString på samtliga objekt i följd och resultatet skrivs ut.

Eftersom en statisk variabeln är gemensam för alla objekt så är det bara det sista värdet på merData som kommer att komma med vid utskriften.

UPPGIFT 3: RÖVARSPRÅKET (6 POÄNG)

Rövarspråket är ett enkelt kodspråk som blev populärt i och med Astrid Lindgrens böcker om Kalle Blomkvist. När man kodar en text med rövarspråket så dubblerar man varje konsonant och placerar ett o mellan dubletterna, till exempel byts b ut mot "bob" och g mot "gog". Vokalerna är oförändrade. "Jag talar rövarspråket" blir alltså "jojagog totalolaror rorövovarorsospoproråkoketot".

Din uppgift är att skriva en metod som tar in en sträng som parameter och som returnerar strängen konverterad enligt ovan.

Två metoder i klassen String som är nödvändiga för att lösa uppgiften är:

- **`char charAt(int index)`**
  - **`Returns the char value at the specified index.`**
- **`int length()`**
  - **`Returns the length of this string.`**

LÖSNINGSFÖRSLAG 1

```
public static String version1(String str) {
   String resultat = "";
   for (int i = 0; i < str.length(); i++) {

      char ch = str.charAt(i);

      switch (ch) {
      case 'b':
      case 'c':
      case 'd':
         // osv för resten av konsonanterna
         resultat += ch + "o" + ch;
         break;
      default:
         // Hanterar såväl vokaler som skiljetecken och blanksteg
         resultat += ch;
      }
   }
   return resultat;
}
```

LÖSNINGSFÖRSLAG 2

Vi kan naturligtvis byta ut switch-satsen mot en if-sats.

```
public static String version2(String str) {
   String resultat = "";
   for (int i = 0; i < str.length(); i++) {
      char ch = str.charAt(i);
      if (ch == 'b' || ch == 'c' || ch == 'd' /* osv */) {
         resultat += ch + "o" + ch;
      } else {
         resultat += ch;
      }
   }
   return resultat;
}
```

LÖSNINGSFÖRSLAG 3

En hjälpmetod kan göra koden enklare att läsa

```
private static boolean konsonant(char ch) {
   return ch == 'b' || ch == 'c' || ch == 'd'; // osv
}

public static String version3(String str) {
   String resultat = "";
   for (int i = 0; i < str.length(); i++) {
      char ch = str.charAt(i);
      if (konsonant(ch)) {
         resultat += ch + "o" + ch;
      } else {
         resultat += ch;
      }
   }
   return resultat;
}
```

LÖSNINGSFÖRSLAG 4

Hjälpmetoden kan vara skriven på många olika sätt, tex med en array:

```
private static final char[] konsonanter = { 'b', 'c', 'd' /* osv */};

private static boolean konsonant(char ch) {
   for (char k : konsonanter) {
      if (ch == k)
         return true;
   }
   return false;
}
```

LÖSNINGSFÖRSLAG 5

Det finns många andra sätt och alternativ för att lösa uppgiften. Ett är att strunta i metoderna vi listade på tentan och byta ut alla konsonanter i strängen.

```
private static final String[] konsonanter = { "b", "c", "d" /* osv */};

public static String version5(String str) {
   for (String k : konsonanter) {
      str = str.replace(k, k + "o" + k);
   }
   return str;
}
```

UPPGIFT 4: JÄMFÖRELSE (6 POÄNG)

Här nedanför ser ni en delar av koden till en klass som representerar ett betyg. Uppgiften är att skriva klart equals-metoden så att det går att jämföra två betygsobjekt med varandra. Objekten är lika om både kurskoden och betyget är lika, annars inte. Du får inte ändra på equals-metodens signatur, den är bestämd eftersom vi överskuggar en metod som finns i Object.

```
public class Betyg {

   private String kurskod;
   private char betyg;

   public Betyg(String kurskod, char betyg) {
      this.kurskod = kurskod;
      this.betyg = betyg;
   }

   // Det är denna metod ni ska skriva klart
   public boolean equals(Object obj) {

   }

   // All övrig kod är ointressant för uppgiften och därför
   // inte medtagen här

}
```

När equals-metoden fungerar som avsett så ska man kunna göra så här:

```
Betyg b1 = new Betyg("OOS", 'C');
Betyg b2 = new Betyg("OOP", 'B');
Betyg b3 = new Betyg("OOP", 'B');

boolean skaVaraFalse = b1.equals(b2);
boolean skaVaraTrue = b2.equals(b3);
```

och de bägge boolska variablerna ska få rätt värde.

LÖSNINGSFÖRSLAG 1

Alla lösningar av denna uppgift måste kontrollera samma saker. Det enda som egentligen skiljer dem åt är hur man väljer att ställa upp dem.

```
public boolean equals(Object obj) {
   if (obj != null && obj instanceof Betyg) {
      // Här vet vi att obj existerar och verkligen är ett betyg och
      // inget annat
      Betyg b = (Betyg) obj;
      return this.kurskod.equals(b.kurskod) && this.betyg == b.betyg;
   } else {
      // Hamnar vi här så var antingen obj null, eller så var det något
      // annat än ett betyg så då kan de inte vara lika
      return false;
   }
}
```

LÖSNINGSFÖRSLAG 2

```
public boolean equals2(Object obj) {
   if (obj == null)
      return false;

   if (!(obj instanceof Betyg))
      return false;

   // Här vet vi att obj verkligen är ett betyg och inget annat
   Betyg b = (Betyg) obj;

   return this.kurskod.equals(b.kurskod) && this.betyg == b.betyg;
}
```

LÖSNINGSFÖRSLAG 3

```
public boolean equals(Object obj) {
   return (obj != null) && (obj instanceof Betyg)
         && this.kurskod.equals(((Betyg) obj).kurskod)
         && (this.betyg == ((Betyg) obj).betyg);
}
```

UPPGIFT 5: PACKA MJÖLK (6 POÄNG)

När jag (Henrik) var student arbetade jag extra i förpackningen på ARLA, något som inspirerat till följande uppgift.

Ett mjölkpaket har ett bäst-före-datum som vi för enkelhetens skull kan anta är ett heltal och en vikt i hela gram. Bägge dessa attribut sätts när mjölkpaketet skapas och kan sedan aldrig ändras. Däremot kan de läsas av när som helst.

Det finns vidare två typer av lastbärare: rullpallar som rymmer 180 mjölkpaket och backar som rymmer 20. Med en lastbärare kan man göra följande saker:

- Lägga till ett mjölkpaket så länge som lastbäraren ännu inte är full.
- Läsa av antalet paket som finns på lastbäraren.
- Läsa av den totala vikten. Här antar vi att en tom rullpall väger 10kg, och en tom back 1kg. Du får själv välja om vikten här ska representeras i gram eller i kilo.
- Läsa av innehållets bäst-före-datum. Detta är det minsta bäst-före-datumet som något av mjölkpaketen i lastbäraren har, eller 0 om lastbäraren är tom.

Din uppgift är att implementera ovanstående modell i Java. Du behöver inte implementera någon ytterligare funktionalitet än den som listas ovan. Du kan alltså strunta i toString-metoder, att man inte kan ta bort paket från lastbärare, etc.

För full poäng ska all funktionalitet vara implementerad och koden följa grundläggande regler för god programmering såsom namngivningskonventioner och användning av lämpliga skyddsnivåer.

LÖSNINGSFÖRSLAG: MJÖLKPAKETEN

```
public class Mjölkpaket {

   // Attributen ska inte kunna ändras, så de är privata.
   // De kan också göras final om man vill.
   private int bästföre;
   private int vikt;

   // Attributen ska sättas när man skapar mjölkpaketet
   public Mjölkpaket(int bästföre, int vikt) {
      this.bästföre = bästföre;
      this.vikt = vikt;
   }

   // Eftersom attributen ska kunna läsas av måste vi ha get-metoder
   // för dem

   public int bästföre() {
      return bästföre;
   }
```

```
   public int vikt() {
      return vikt;
   }

}
```

LÖSNINGSFÖRSLAG 1

```
/**
 * En version av lastbärare som använder abstrakta metoder för att hålla
 * reda på hur många paket som får plats och hur mycket lastbäraren väger
 * Den totala vikten anges i gram.
 */
public abstract class LastbärareV1 {
   private ArrayList<Mjölkpaket> paket = new ArrayList<>();

   protected abstract int maxPaket();

   public void läggTill(Mjölkpaket p) {
      if (antalPaket() < maxPaket()) {
         paket.add(p);
      }
   }

   public int antalPaket() {
      return paket.size();
   }

   protected abstract int lastbärarvikt();

   public int totalvikt() {
      int total = lastbärarvikt();
      for (Mjölkpaket p : paket) {
         total += p.vikt();
      }
      return total;
   }

   public int bästföre() {
      if (antalPaket() == 0) {
         // Här saknas det information i uppgiften om vad som ska
         // returneras, så vi gör ett antagande och returnerar -1, en
         // vanlig felkod.
         return -1;
      } else {
         int sämstDatum = paket.get(0).bästföre();
         for (Mjölkpaket p : paket) {
```

```
            if (p.bästföre() < sämstDatum) {
               sämstDatum = p.bästföre();
            }
         }
         return sämstDatum;
      }
   }

}

public class RullpallV1 extends LastbärareV1 {

   protected int maxPaket() {
      return 180;
   }

   protected int lastbärarvikt() {
      return 10000;
   }

}

public class BackV1 extends LastbärareV1 {

   protected int maxPaket() {
      return 20;
   }

   protected int lastbärarvikt() {
      return 1000;
   }

}
```

LÖSNINGSFÖRSLAG 2

```
/**
 * En version av lastbärare som använder variabler satta i konstruktorn
 * för att hålla reda på hur många paket som får plats och hur mycket
 * lastbäraren väger. Den totala vikten anges i gram.
 */
public abstract class LastbärareV2 {
   private ArrayList<Mjölkpaket> paket = new ArrayList<>();

   private int maxPaket;
   private int lastbärarvikt;

   public LastbärareV2(int max, int vikt) {
      maxPaket = max;
      lastbärarvikt = vikt;
   }

   public void läggTill(Mjölkpaket p) {
      if (antalPaket() < maxPaket) {
         paket.add(p);
      }
   }

   public int antalPaket() {
      return paket.size();
   }

   public int totalvikt() {
      int total = lastbärarvikt;
      for (Mjölkpaket p : paket) {
         total += p.vikt();
      }
      return total;
   }

   public int bästföre() {
      if (antalPaket() == 0) {
         // Här saknas det information i uppgiften om vad som ska
         // returneras, så vi gör ett antagande och returnerar -1, en
         // vanlig felkod.
         return -1;
      } else {
         int sämstDatum = paket.get(0).bästföre();
         for (Mjölkpaket p : paket) {
            if (p.bästföre() < sämstDatum) {
               sämstDatum = p.bästföre();
            }
```

```
            }
            return sämstDatum;
        }
    }

}

public class RullpallV2 extends LastbärareV2 {

    public RullpallV2() {
        super(180, 10000);
    }

}

public class BackV2 extends LastbärareV2 {

    public BackV2() {
        super(20, 1000);
    }

}
```