TENTAMEN OOP 2014-03-15

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)

```
String s1 = 5 + "+" + 2 + " är " + 5 + 2;
String s2 = 5 + "/" + 2 + " är " + 5 / 2;
s1.toUpperCase();
s2.toUpperCase();
System.out.println(s1);
System.out.println(s2);
```

KORREKT SVAR

```
5+2 är 52
5/2 är 2
```

VARFÖR BLIR SVARET VAD DET BLIR?

Plussar man ihop en sträng och en integer får man en sträng. Detta utnyttjas genomgående när vi skapar s1. Att 5/2 faktiskt utför en division beror på att du inte kan dividera strängar, utan det måste vara frågan om (heltals)division. Att resultatet blir 2 och inte 2,5 beror på att det är två heltal som divideras. Då klipps ju resultatet vid decimalkommat.

De bägge raderna med toUpperCase har ingen effekt alls eftersom deras uppgift är att returnera en kopia av strängen med enbart stora bokstäver, inte att förändra strängen. Kom ihåg: strängar i Java kan aldrig förändras, man arbetar alltid med kopior.

DELUPPGIFT B (2 POÄNG)

```
for (int tal = 6; tal > 0; tal -= 2) {
   switch (tal) {
   case 1:
   case 2:
      System.out.print(tal);
   case 3:
   case 4:
      System.out.print(tal);
   default:
      System.out.print(tal);
   }
}
```

KORREKT SVAR

644222

VARFÖR BLIR SVARET VAD DET BLIR?

Loopen kommer att gå tre varv. I det första är tal 6, i det andra 4 och i det tredje 2. I varje varv använder vi tal för att välja case i switch-satsen. Eftersom det inte finns några break så kommer alla print-satser som står efter det case vi hamnat i att köras.

DELUPPGIFT C (2 POÄNG)

```
int[] arr = { 3, 1, 4, 1, 5 };
for (int i = 0; i < arr.length; i++) {
   if (arr[i] < arr.length) {
      arr[arr[i]] = arr[i];
   }
}
for (int x : arr) {
   System.out.print(x);
}
```

KORREKT SVAR

31434

VARFÖR BLIR SVARET VAD DET BLIR?

Den första for-loopen ändrar arrayen, och den andra skriver sedan ut den. Ändringarna som görs framgår av nedanstående tabell:

i	arr[i]	Resultat	Kommentar
0	3	3, 1, 4, **3**, 5	
1	1	3, **1**, 4, 3, 5	1 ersätts med 1
2	4	3, 1, 4, 3, **4**	
3	3	3, 1, 4, **3**, 4	3 ersätts med 3
4	4	3, 1, 4, 3, **4**	4 ersätts med 4

UPPGIFT 2: KODFÖRSTÅELSE KLASSER OCH OBJEKT (6 POÄNG)

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

```
import java.util.ArrayList;

class MainClass {
   public static void main(String[] args) {
      ArrayList<MellanKlass> lista = new ArrayList<>();

      lista.add(new MellanKlass());
      lista.add(new SubKlass());
      lista.add(new SubKlass(1));

      for (MellanKlass mk : lista) {
         System.out.print(mk.japp());
      }
   }
}
```

(Övriga klasser finns på nästa sida.)

```
abstract class SuperKlass {

   protected int värde;

   public SuperKlass() {
      this(2);
   }

   public SuperKlass(int tal) {
      värde = tal;
   }

   public int japp() {
      return värde;
   }

}

class MellanKlass extends SuperKlass {

   public int daim() {
      return 3;
   }

   public int japp() {
      return daim() * 10 + super.japp();
   }
}

class SubKlass extends MellanKlass {

   private static int x = 4;

   public SubKlass() {
      this(5);
   }

   public SubKlass(int tal) {
      x += tal;
   }

   public int daim() {
      return x;
   }

}
```

KORREKT SVAR

32102102

VARFÖR BLIR SVARET VAD DET BLIR?

Vi skapar först en instans av MellanKlass med hjälp av den automatiskt genererade parameterlösa konstruktorn som finns i den. Denna konstruktor anropar SuperKlass paramterlösa konstruktor som i sin tur anropar SuperKlass andra konstruktor med argumentet 2. Detta tal lagras i variabeln värde.

Därefter skapar vi en instans av SubKlass med hjälp av den parameterlösa konstruktorn som finns i den. Denna konstruktor anropar den andra konstruktorn i SubKlass med argumentet 5. Därefter anropas MellanKlass automatiskt genererade parameterlösa konstruktor som i sin tur anropar SuperKlass paramterlösa konstruktor som i sin tur anropar SuperKlass andra konstruktor med argumentet 2. Detta tal lagras i variabeln värde för det nya objektet. Slutligen återvänder vi till SubKlass andra konstruktor och ökar värdet på den den statiska variabeln x med 5 som vi fick in via parametern.

Slutligen skapar vi en instans av SubKlass med hjälp av den andra konstruktorn och argumentet 1. Det som skiljer skapandet av denna instans från den förra är egentligen bara att vi går direkt på den andra konstruktorn och att vi ökar den statiska variabeln x med 1 så att den nu är 10. Kom ihåg att statiska variabler tillhör klassen och alltså delas av alla instanser.

Vi har nu tre objekt:

- En instans av MellanKlass med värde 2
- En instans av SubKlass med värde 2
- En instans till av SubKlass med värde 2

Dessutom har vi en statisk variabel i klassen SubKlass: x, som har värdet 1.

Nu går vi igenom listan vi placerat objekten i och anropar metoden japp på dem. Eftersom det första objektet är av typen MellanKlass är det japp i MellanKlass som körs. Denna metod anropar daim (i MellanKlass) och superklassens version av japp, och returnerar 3*10+2 = 32.

För de bägge SubKlass-objekten händer exakt samma sak. japp i MellanKlass anropas. Den i sin tur anropar daim i SubKlass eftersom objekten är av typen SubKlass och superklassens version av japp, och returnerar 10*10+2 =102.

UPPGIFT 3: LIKA TALSEKVENSER (6 POÄNG)

Skriv ett program som läser in tio heltal från användaren och som därefter berättar om de första fem var samma som de sista fem utan hänsyn till ordningen. Om användaren till exempel skriver in:

2 1 1 0 88 och **0 1 88 2 1**

ska programmet alltså säga att bägge uppsättningarna tal är lika eftersom bägge uppsättningarna tal innehåller precis samma siffror: 0, 1, 1, 2 och 88.

Om användaren istället skriver in

0 0 1 1 1 och **0 1 0 1 0**

så är de däremot inte lika eftersom det finns två nollor och tre ettor i den första uppsättningen och tre nollor och två ettor i den andra.

Du får själv välja hur användaren skriver in talen, det spelar ingen roll för uppgiften. Du kan också förutsätta att användaren alltid skriver in tal, så det behövs ingen felhantering.

LÖSNINGSFÖRSLAG 1

```
import java.util.Arrays;
import java.util.Scanner;

public class LikaTal1 {

   public static void main(String[] args) {

      Scanner s = new Scanner(System.in);

      System.out.println("Skriv fem tal...");

      int[] femTal = new int[5];
      for(int i =0; i<5 ; i++){
         femTal[i]=s.nextInt();
      }

      System.out.println("Skriv fem tal till...");

      int[] femTalTill = new int[5];
      for(int i =0; i<5 ; i++){
         femTalTill[i]=s.nextInt();
      }

      Arrays.sort(femTal);
      Arrays.sort(femTalTill);

      boolean lika=true;
```

```
      for(int i =0; i<5 ; i++){
         if(femTal[i]!=femTalTill[i]){
            lika=false;
         }
      }

      if(lika){
         System.out.println("De bägge uppsättningarna tal var lika");
      }else{
         System.out.println("De bägge uppsättningarna tal var inte
lika");
      }
   }

}
```

LÖSNINGSFÖRSLAG 2

```
import java.util.Arrays;
import java.util.Scanner;

public class LikaTal2 {

   private static Scanner s = new Scanner(System.in);

   private static int[] läsTal(){
      int[] tal = new int[5];
      for(int i =0; i<5 ; i++){
         tal[i]=s.nextInt();
      }
      return tal;
   }

   private static boolean lika(int[] tal1, int[] tal2){
      Arrays.sort(tal1);
      Arrays.sort(tal2);
      for(int i =0; i<5 ; i++){
         if(tal1[i]!=tal2[i]){
            return false;
         }
      }
      return true;
   }

   public static void main(String[] args) {
      System.out.println("Skriv fem tal...");
      int[] femTal = läsTal();
```

```
      System.out.println("Skriv fem tal till...");
      int[] femTalTill = läsTal();


      if(lika(femTal, femTalTill)){
         System.out.println("De bägge uppsättningarna tal var lika");
      }else{
         System.out.println("De bägge uppsättningarna tal var inte
lika");
      }
   }

}
```

UPPGIFT 4: (6 POÄNG)

I en del affärer, bland annat ICA i Kista centrum, finns det elektroniska prislappar på hyllorna, små strömsnåla skärmar som visar priset för en vara.

Vi vill nu att du skriver en klass som representerar en sådan elektronisk prislapp. Det är en klass som innehåller uppgifter om namnet och priset på en vara. För enkelhetens skull antar vi att priset alltid är i hela kronor. Namnet och priset ska alltid sättas när man skapar en prislapp och ska sen gå att ändra och läsa av. Dock ska det inte gå att sätta priset till lägre än 0kr på något sätt. Om man försöker ska priset sättas till 0kr.

Förutom namn och priset ska det också finnas information om priset är per styck eller om det är per kilo vilket representeras med hjälp av en enum:

```
enum Varutyp{
   STYCK, VIKT
}
```

Detta är viktigt vid utskrift, något som klassen naturligtvis ska klara på Javas vedertagna sätt.[1] Om priset på choklad är 14kr och priset är per kilo ska utskriften bli ”Choklad 14kr/kilo”. Om priset är per styck så ska utskriften istället bli ”Choklad 14kr/styck”. Vid skapandet av objekt är det frivilligt om man vill ange vilken enhet det rör sig om, styck eller kilo. Om man inte anger enheten ska den per default sättas till per styck. Även om priset är per styck eller per kilo ska kunna läsas av och ändras.

Ett exempel[2] hur klassen ska kunna användas:

```
Prislapp p1 = new Prislapp( ”Choklad”, 14 );
Prislapp p2 = new Prislapp( ”Mjölk”, 13, Varutyp.STYCK );
Prislapp p3 = new Prislapp( ”Äpplen”, 10, Varutyp.VIKT );
System.out.println(p1);
System.out.println(p2);
System.out.println(p3);
```

Detta ger utskriften

```
Choklad 14kr/st
Mjölk 13kr/st
Äpplen 10kr/kilo
```

[1] Det ingår i uppgiften att veta vad detta innebär.
[2] Observera att exemplet bara visar ett par av de saker som klassen ska kunna göra. Alla delar som nämns i uppgiften ska vara med och fungera för full poäng.

LÖSNINGSFÖRSLAG

```
public class Prislapp {

   private String namn;    private int pris;     private Varutyp typ;

   public Prislapp(String namn, int pris) {
      this(namn, pris, Varutyp.STYCK);
   }

   public Prislapp(String namn, int pris, Varutyp typ) {
      this.namn = namn;
      setPris(pris);
      this.typ = typ;
   }

   public String getNamn() {
      return namn;
   }

   public void setNamn(String nyttNamn) {
      this.namn = nyttNamn;
   }

   public int getPris() {
      return pris;
   }

   public void setPris(int nyttPris) {
      if (nyttPris < 0)
         this.pris = 0;
      else
         this.pris = nyttPris;
   }

   public Varutyp getTyp() {
      return typ;
   }

   public void setTyp(Varutyp nyTyp) {
      this.typ = nyTyp;
   }

   public String toString() {
      return namn + " " + pris + "kr/" + (typ == Varutyp.STYCK ? "st" :
"kg");
   }
}
```

ALTERNATIV TOSTRING

```
public String toString() {
   String s = namn + " " + pris + "kr/";
   if (typ == Varutyp.STYCK)
      s += "st";
   else
      s += "kg";
   return s;
}
```

UPPGIFT 5: BTD (6 POÄNG)

Bloons Tower Defence (BTD) är en serie webbspel från Ninja Kiwi där man som spelare ska hindra ballonger av olika typ från att ta sig från ena änden av en bana till den andra genom att placera ut torn som ska skjuta sönder ballongerna. Ballongerna är konstruerade så att de innehåller andra ballonger som släpps lösa när man förstör den omkringliggande ballongen.[3]

Nedanstående klasser är ett försök att modellera ballongerna som används i dessa spel. Din uppgift är att skriva klart klasserna så att metoden hit fungerar som avsett för röda, gula och svarta ballonger. Du får fritt lägga till metoder, konstruktorer, variabler, etc. som du tycker behövs för uppgiften. Om du vill får du också byta ut arrayen som metoden **`hit`** returnerar mot en ArrayList.

```
abstract class Bloon {

   protected int life;

   public int getLife(){ return life; }

   /*
    * Drar bort ett liv från ballongen. Om antalet liv kvar är större än
    * noll, eller om ballongen inte innehåller några andra ballonger, så
    * returnerar metoden en tom array. Om antalet liv är noll returneras
    * en array med de ballonger som finns inuti i denna.
    */
   abstract Bloon[] hit();
}


// Har ett liv och innehåller inga andra ballonger
class Red extends Bloon {

}

// Har fem liv och innehåller två röda ballonger
class Yellow extends Bloon {

}

// Har tio liv och innehåller två gula ballonger
class Black extends Bloon {

}
```

[3] Allt som står i detta stycke är ren bakgrundsinformation som bara finns med för att motivera de annars meningslösa klasserna ni ska skriva klart. Tornen, banorna etc. har alltså inget med uppgiften som sådan att göra, och kommer inte att förekomma i lösningen.

LÖSNINGSFÖRSLAG

```
abstract class Bloon {

   // Eftersom vi ofta måste returnera en tom array så kan det vara bra
   // att ha en konstant sådan så att vi inte behöver skapa en hela tiden
   // Denna konstant är inte på något vis nödvändig, bara praktisk.
   protected static final Bloon[] EMPTY = {};

   protected int life;

   public Bloon(int life) {
      this.life = life;
   }

   public int getLife() {
      return life;
   }

   // Hjälpmetod som drar bort ett liv från ballongen (om ballongen har
   // några liv kvar vill säga) och som returnerar om ballongen har några
   // liv kvar eller inte.
   protected boolean removeLife() {
      if (life > 0) {
         life--;
      }
      return life > 0;
   }

   /*
    * Drar bort ett liv från ballongen. Om antalet liv kvar är större än
    * noll, eller om ballongen inte innehåller några andra ballonger, så
    * returnerar metoden en tom array. Om antalet liv är noll returneras
    * en array med de ballonger som finns inuti i denna.
    */
   abstract Bloon[] hit();
}

// Har ett liv och innehåller inga andra ballonger
class Red extends Bloon {

   public Red() {
      super(1);
   }

   Bloon[] hit() {
```

```
      // Här behöver vi ingen kontroll eftersom ballongen bara har ett
liv
      removeLife();
      return EMPTY;
   }

}

// Har fem liv och innehåller två röda ballonger
class Yellow extends Bloon {
   public Yellow() {
      super(5);
   }

   Bloon[] hit() {
      if (removeLife()) {
         return EMPTY;
      } else {
         Bloon[] inside = new Bloon[2];
         inside[0] = new Red();
         inside[1] = new Red();
         return inside;
      }
   }

}

// Har tio liv och innehåller två gula ballonger
class Black extends Bloon {
   public Black() {
      super(10);
   }

   Bloon[] hit() {
      if (removeLife()) {
         return EMPTY;
      } else {
         Bloon[] inside = new Bloon[2];
         inside[0] = new Yellow();
         inside[1] = new Yellow();
         return inside;
      }
   }

}
```