Omtentamen för TDA540
Objektorienterad Programmering

Institutionen för Datavetenskap
CTH HT-16, TDA540

Dag: 2017-08-24, *Tid:* 14.00-18.00

Ansvarig:	Alex Gerdes
Examinator:	Carlo A. Furia
Förfrågningar:	Alex Gerdes (alexg@chalmers.se, 031 772 6154)
Resultat:	Erhålls via Ladok
Betygsgränser:	3:a 24 poäng 4:a 36 poäng 5:a 48 poäng max 60 poäng
Siffror inom parentes:	Anger maximal poäng på uppgiften
Granskning:	Tentamen kan granskas på studieexpeditionen. Vid eventuella åsikter om rättningen eposta och ange noggrant vad du anser är fel så återkommer vi.
Hjälpmedel:	Cay Horstmann: *Java for everyone* eller Jan Skansholm: *Java direkt med Swing* Understrykningar och smärre förtydligande noteringar får finnas.
Var vänlig och:	Skriv tydligt och disponera papperet på lämpligt sätt. Börja varje uppgift på nytt blad. Skriv ej på baksidan av papperet.
Observera:	Uppgifterna är ej ordnade efter svårighetsgrad. Titta därför igenom hela tentamen innan du börjar skriva. Alla program skall vara väl strukturerade, lättöverskådliga och enkla att förstå. Indentera programkoden! Vid rättning av uppgifter där programkod ingår bedöms principiella fel allvarligare än smärre språkfel.

Lycka till!

Uppgift 1

(4 poäng)

Vad avses med?

a) Abstrakt klass (abstract class)

b) Arv (inheritance)

c) Undantag (exception)

d) Variabel (variable)

Förklara med en eller ett par meningar, du får gärna förtydliga med en skiss eller kod.

Uppgift 2

(4 poäng)

Vilka av raderna 1-11 nedan kompilerar ej? Motiverar varför!

```
1   class Number {
2     private int num;
3
4     public Nummer(int n) {
5       num = n;
6     }
7
8     public int getNumber() {
9       return n;
10    }
11  }
```

Uppgift 3

(5 poäng)

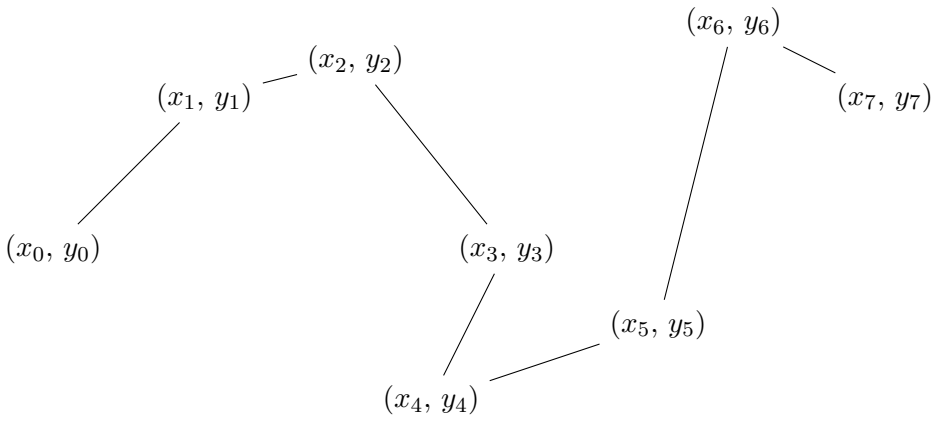
Exponentialfunktionen används ofta i matematiken och skrivs som `exp(x)` i de flesta programspråk. Man kan beräkna den med en potensserie:

$$exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \dots$$

I ovanstående betyder $n!$ att det är n-fakultet, dvs multiplikation av alla heltal från 1 till och med n. Skriv en metod `exp` som gör en beräkning enligt ovanstående funktion. Metoden `exp` tar x som argument och ska ta med de första 100 termer från potensserien. Observera att det är inte tillåtit att använda metoderna `Math.pow` eller `Math.exp` samt `^` operatorn.

Uppgift 4 (12 poäng)

Ett program behöver hantera "vägar" i planet som i nedanstående exempel:



Vägen bestäms i exemplet av punkterna $(x_0, y_0), (x_1, y_1), \ldots, (x_7, y_7)$; antalet punkter i en väg är förstås inte alltid åtta.

a) Definiera en klass `Path` vars instanser representerar en sådan väg. Som hjälpklass ska du definiera en klass `Point` som representerar en punkt som innehåller en x och y koordinat. Konstruktorn i `Path` ska ta en punkt (startpunkten) som argument.

b) Vidare ska det finnas en metod `addPoint`, som också tar en punkt som argument och lägger till denna punkt som ny slutpunkt, observera att den får inte vara samma som den befintliga slutpunkten.

c) Det ska finnas en metod `getPoints` som returnerar hela vägen, dvs alla punkterna, i någon form. Du får själv välja en lämplig resultattyp; det kan vara en lista eller ett fält eller kanske något annat.

d) Överskugg `toString` metoden så att en väg skrivs ut på ett lämpligt sätt.

e) Lägg till en metod `boolean loops()` i `Path` klassen som kollar om vägen loopar, dvs när vi följer vägen kommer vi tillbaka till någon punkt där vi har varit förut.

Uppgift 5 (3 poäng)

Betrakta metoden `main()` nedan och ange vad skrivs ut när den körs.

```
public class Uppgift5 {
  public static void setElement(int[] xs, int i, int n) {
    if (i < xs.length) {
      xs[i] = n;
    } else {
      System.out.println("Index out of bounds!");
    }
  }

  public static void setList(int[] xs, int n) {
    int[] ys = new int[xs.length];

    for (int i = 0; i < ys.length; i++) {
      ys[i] = n;
    }
    xs = ys;
  }

  public static void printList(int[] xs) {
    for (int x : xs)
      System.out.println(x);
  }

  public static void main(String[] args) {
    int[] list = {1, 3, 3, 7};
    setElement(list, 3, 8);
    printList(list);
    setList(list, 42);
    printList(list);
  }
}
```

Uppgift 6

a) Skriv en metod med följande signatur:

```
static int digit(int n, int k)
```

som returnerar siffran på position `k` i talet `n`, räknad från positionen längst till höger. Om `n` har mindre än `k` siffror då returneras `-1`. Till exempel:

- `digits(372, 1)` returnerar `2`,
- `digits(372, 3)` returnerar `3`, och
- `digits(372, 4)` returnerar `-1`.

Du får anta att både `n` och `k` är positiva heltal. (5 poäng)

b) Skriv en metod `increasing` med följande signatur:

```
static boolean increasing(int n)
```

som kontrollerar om siffrorna i talet `n` är i stigande ordning när man räknar upp från positionen längst till höger, dvs första siffran (längst till höger) är mindre eller lika med den andra siffran, vilken in sin tur är mindre eller lika med den tredje siffran, och så vidare. Till exempel:

- `increasing(372)` returnerar `false`,
- `increasing(743)` returnerar `true`.

Du får anta att `n` är ett positivt heltal och behöver inte kontrollera detta. Implementationen av `increasing` skall anropa `digit` och gå genom siffrorna i `n`. (5 poäng)

Uppgift 7

Betrakta följande klass- och gränssnittsdeklarationer:

```
interface Immutable<G> {
   boolean has(G v);
}

interface Collection<G> extends Immutable<G> {
   void put(G v);
}

class Sequence<G> implements Collection<G> {
   ArrayList<G> elements = new ArrayList<>();
   public boolean has(G v) { return elements.contains(v); }
   public void put(G v) { elements.add(v); }
}

class IntSequence extends Sequence<Integer> {
}

class IntList implements Immutable<Integer> {
   Integer[] elements = {5, 10, 20, 30, 36, 43};
   public boolean has(Integer v) {
      for (int i = 0; i < elements.length; i++) if (elements[i] == v) return true;
      return false;
   }
}
```

a) Förklara för varje nedanstående sats om den blir accepterad eller resulterar i ett kompileringsfel, om det är fel förklara varför. (10 poäng)

1. `Immutable<Integer> x1 = new Immutable<Integer>();`
2. `Collection<Integer> x2 = new Sequence<Integer>();`
3. `Immutable<Integer> x3 = new Sequence<Integer>();`
4. `Sequence<Integer> x4 = new Sequence<Integer>();`
5. `IntSequence x5 = new Sequence<Integer>();`
6. `Sequence<Integer> x6 = new IntSequence();`
7. `IntList x7 = new Sequence<Integer>();`
8. `Immutable<Integer> x8 = new IntList();`
9. `Sequence<Double> x9 = new Sequence<Double>();`
10. `Sequence<Integer> x10 = new Sequence<Double>();`

b) Vad skriver följande satser ut? (2 poäng)

1. `System.out.println(new IntList().has(20));`
2. `System.out.println(new IntSequence().has(20));`
3. `System.out.println(((Immutable<Integer>)new IntSequence()).has(20));`
4. `System.out.println(((Immutable<Integer>)new IntSequence()).put(20));`

Uppgift 8

I den här uppgift ska vi använda instanser av nedanstående klassen `Sequence`, som representerar sekvenser av element som har en godtycklig generisk typ `G`. Klassen använder ett privat fält (array), för att spara alla element, som blir initialiserat i konstruktorn.

```
class Sequence<G> {
  private final G[] elements;

  public Sequence(G[] elements) {
     this.elements = elements;
  }
}
```

En elements *multiplicitet* (engelska: multiplicity) i en sekvens är antalet förekomster av elementet i sekvensen. Till exempel, multipliciteten av 3 i $1, 2, 3, 4, 3, 2, 1$ är 2, och multipliciteten av 5 is samma sekvens är 0.

a) Utöka klassen `Sequence` med en metod `multip`:

```
int multip(G x)
```

som returnerar multipliciteten av `x` i `this` objektets sekvens. (4 poäng)

Betrakta nu två sekvenser $s = x_0, x_1, \ldots, x_n$ och $t = y_0, y_1, \ldots, y_m$. s är en *permutation* av t om multipliciteten av *varje element* i s och t är detsamma i s så som i t. Till exempel, $s = 1, 2, 2, 3$ är en permutation av $t = 2, 1, 3, 2$, men inte av $u = 1, 2, 3$ för multipliciteten av 2 i s är 2 medan det är 1 i u.

b) Utöka klassen `Sequence` med en metod `permutation`:

```
boolean permutation(Sequence<G> t)
```

som returnerar `true` om `this` objektets sekvens är en permutation av argumentets (`t`) sekvens och annars `false`. Implementationen av `permutation` skall följa ovanstående definition av permutation och använda `multip` för att beräkna elementens multiplicitet. (6 poäng)