NTNU
Institutt for datateknikk
og informasjonsvitenskap

Eksamensoppgave i

TDT4100 – Objektorientert programmering

Fredag 19. august 2011, kl. 09:00 - 13:00

Oppgaven er utarbeidet av faglærer Hallvard Trætteberg og kvalitetssikrer Trond Aalberg.
Kontaktperson under eksamen Hallvard Trætteberg (mobil 918 97263)

Språkform: Bokmål
Tillatte hjelpemidler: C
Én valgfri lærebok i Java er tillatt.
Bestemt, enkel kalkulator tillatt.

Sensurfrist: Fredag 9. september

Les oppgaveteksten nøye. Finn ut hva det spørres etter i hver oppgave.

Dersom du mener at opplysninger mangler i en oppgaveformulering gjør kort rede for de antagelser og forutsetninger som du finner det nødvendig å gjøre.

Del 1 – Metoder (30%)

”Tre små trøndere på Høili plass” er en barnesang som synges litt spesielt. Første vers synges med ’a’-lyd for alle vokalene, med andre ord ”Tra sma trandara pa…”, andre vers synges med ’e’-lyd for alle vokalene og så videre for alle vokalene i alfabetet. I denne oppgaven skal du lage en metode for å skrive ut sanger på denne formen, med tilhørende hjelpemetoder. Du kan anta at alle metodene og evt. felt du ønsker å innføre samles i en felles klasse kalt **Song**. For alle deloppgavene så belønnes du for å bruke (hjelpe)metoder fra tidligere deloppgaver.

a) Skriv en metode **isVowel(char c)** som returnerer **true** dersom **c** er en vokal og **false** ellers. Både store og små bokstaver må håndteres.

```
private static String vowels = "aeiouyæøå";

public static boolean isVowel(char c) {
      return vowels.indexOf(Character.toLowerCase(c)) >= 0;
}
```

b) Skriv en metode **String computeVerse(String originalVerse, char v)** som tar inn originalverset og en vokal **v** som parametre. Metoden skal returnerer verset, med vokalene i originalverset erstattet med den angitte vokalen **v**. Dersom **computeVerse** kalles med **”Alfabet”** og **’i’** som argument, så skal **”Ilfibit”** returneres. Merk at også her må store og små bokstaver håndteres riktig. Dersom **v** ikke er en vokal så skal det kastes et passende unntak.

```
public static String computeVerse(String originalVerse, char v) {
      if (! isVowel(v)) {
            throw new IllegalArgumentException(v + " is not a vowel");
      }
      String verse = "";
      for (int i = 0; i < originalVerse.length(); i++) {
            char c = originalVerse.charAt(i);
            if (! isVowel(c)) {
                  verse += c;
            } else if (Character.isUpperCase(c)) {
                  verse += Character.toUpperCase(v);
            } else {
                  verse += Character.toLowerCase(v);
            }
      }
      return verse;
}
```

c) Skriv en metode **writeSong(String originalVerse)** som tar inn originalverset som parameter og skriver ut alle versene, hvor hvert vers har alle vokalene erstattet med en bestemt vokal. Det skal altså skrives ut ett vers for hver vokal i alfabetet. Dersom **writeSong** kalles med **”Alfabet”** som argument, så skal **”Alfabat\nElfebet\nIlfibit\n…\nYlfybyt”** skrives ut (**\n** er et linjeskift mellom versene).

```
public static void writeSong(String originalVerse) {
      for (int i = 0; i < vowels.length(); i++) {
            System.out.println(computeVerse(originalVerse, vowels.charAt(i)));
```

```
        }
}
```

d) Du skal gjøre det mulig å skrive ut en sang med vers for et bestemt/begrenset sett med vokaler. Dette skal gjøres ved at **Song**-klassen instansieres med originalverset og **writeSong**-metoden kalles med settet av vokaler. Dersom en utfører **new Song("Alfabet").writeSong("aei")**, så skal **"Alfabat\nElfebet\nIlfibit"** skrives ut (**\n** er fortsatt linjeskift). Forklar med tekst og kode hvilke endringer som må gjøres for at dette skal være mulig.

En må introdusere et String-felt for originalverset. Konstruktøren må ta inn originalverset og sette feltet. Metodene **computeVerse** og **writeSong** skal ikke lenger ta inn originalverset, men lese det tilsvarende feltet. Da kan de heller ikke lenger være deklarert med **static**. **writeSong**-metoden må iterere over vokalene i argumentet, ikke alle vokalene i alfabetet.

Del 2 – Klasser (35%)

Du skal realisere et rating/poengsystem for et nettsted hvor personer kan publisere artikler/innlegg. Både personer og innlegg har en rating. Ratingen er et tall mellom 1 og 5 og starter som 1. De som leser innleggene kan gi 1 til 5 poeng til innleggene og/eller forfatterne for å øke eller redusere deres rating. Ratingen til både personer og innlegg er til enhver tid gjennomsnittet av poengene de har fått. Dersom et innlegg har fått totalt 10 poeng på 5 poenggivinger, vil innlegget ha 10 / 5 = 2 som rating.

Personer med høy rating regnes som viktigere enn de med lav rating og poengene som gis vektes derfor på følgende måte: Dersom en person med rating R gir P poeng, så regnes det som R poenggivinger à P poeng. Dersom innlegget i eksemplet over, med 10 poeng på 5 poenggivinger, får 4 poeng (P) fra en person med 1 som rating (R), så vil den nye ratingen være (10 + P*R) / (5+R) = 14/6 = 2,333.... Ratingen øker altså fra 2 til 2,333... fordi samlet antall ratingpoeng øker fra 10 til 14 og antall poenggivinger fra 5 til 6. Dersom innlegget istedenfor får 4 poeng (P) fra en person med 3 som rating (R), så vil den nye ratingen være (10 + P*R) / (5+R) = 22/8 = 2,75. Ratingen øker altså fra 2 til 2,75 fordi samlet antall ratingpoeng øker fra 10 til 22 og antall poenggivinger fra 5 til 8.

Rating av personer håndteres på samme måte som for innlegg.

a) Implementer klassen **Person** med *felt* og *innkapslingsmetoder* for navn og nødvendige *felt* for å håndtere ratingen. For å endre ratingen skal **Person**-klassen kun ha én metode, **receivePoints(int points, Person giver)**, som kan brukes når personen **giver** gir **points** poeng til **Person**-objektet som metoden kalles på. Dersom **randi** og **jon** refererer til hvert sitt **Person**-objekt, så skal altså kallet **randi.receivePoints(5, jon)** oppdatere **randi** sin rating iht. de 5 poengene og **jon** sin rating, som beskrevet over. Husk å validere argumentene!

b) Du skal implementere en **Submission**-klasse (innlegg) og tenker at du kan spare kode ved å flytte kode for å håndtere rating til en egen klasse og (gjen)bruke denne vha. arv. Implementer en slik løsning.

```
public class Rating {

      private double pointSum = 1, weightSum = 1;

      public void receivePoints(int points, Person rater) {
            if (points < 1 || points > 5) {
```

```
                throw new IllegalArgumentException("points must be between 1
and 5, but was " + points);
            }
            double raterWeight = rater.getRating();
            pointSum += raterWeight * points;
            weightSum += raterWeight;
      }

      public double getRating() {
            return pointSum / weightSum;
      }
}
```

Person og Submission arver fra Rating.

c) Delegering er en fleksibel teknikk for å (gjen)bruke nyttig funksjonalitet i andre klasser. Vis hvordan kode for å håndtere rating kan samles i en egen klasse, som klassene **Person** og **Submission** kan delegere til, istedenfor å bruke arv.

Rating blir som over.

Person:

```
public class Person {

      private Rating rating = new Rating();

      public void receivePoints(int points, Person rater) {
            rating.receivePoints(points, rater);
      }

      public double getRating() {
            return rating.getRating();
      }
}
```

Submission blir tilsvarende.

Del 3 – Testing (15%)

a) Forklar med tekst og kode hvordan du vil teste hver av metodene i klassen **Song** fra oppgave 1 a-c.

```
public class SongTest extends TestCase {

      private static String vowels = "aeiouyæøå", consonants =
"bcdfghjklmnpqrstvwxz";

      public void testIsVowel() {
            for (int i = 0; i < vowels.length(); i++) {
                  assertTrue(Song.isVowel(vowels.charAt(i)));
            }
```

```
            for (int i = 0; i < consonants.length(); i++) {
                  assertFalse(Song.isVowel(consonants.charAt(i)));
            }
      }

      public void testComputeVerse() {
            assertEquals("Ilfibit", Song.computeVerse("Alfabet", 'i'));
      }

      public void testWriteSong() {
            Song.writeSong("Alfabet");
      }
}
```

b) Forklar med tekst og kode hvordan du vil teste **Song**-klassen fra oppgave 1 d).

Her kan en bruke samme test-strategi, men utføre kallene iht. det nye designet.

Del 4 – Java-forståelse (20%)

a) I Java må alle deklarasjoner av felt, parametre og variabler inneholde en type, f.eks. **int a**. Hva er hensikten med deklarasjon av typer i et programmeringsspråk?

Hensikten med typer er å kunne begrense/sjekke hvilke operasjoner som er lovlige ved kompilering/i editoren. Uten typer kan en f.eks. ikke sjekke om a * b er lovlig eller ikke.

b) Hva skrives ut av følgende kode? Forklar hvorfor!
System.out.println(”Hallvard sin alder er ” + 4 * 10 + 4);

Utskriften blir ”Hallvard sin alder er 404”. Pga. operator-presedensen beregnes først 4 * 10 (= 40) og så utføres +’ene fra venstre mot høyre. String + int + int blir tolket som ((String + int) + int), og String + int utføres ved at int’en gjøres om til en String og legges til (append’es). Intensjonen var antageligvis å legge sammen 40 og 4 som tall, og for å få til det må en legge parenteser rundt slik: **”Hallvard sin alder er ” + (4 * 10 + 4)**.

c) Anta følgende tre varianter av klassen **A**. I hvert tilfelle utføres **new A()** og en får utskriften som vist under hver variant, hhv. ”1”, ”2” og ”1”. Forklar hvorfor!

```
public class A {
  private int i = 0;
  private int foo() {
    i = i + 1;
    return i;
  }
  public A() {
    if (foo() != 0 || foo() == 0) {
      System.out.println(i);
    }
  }
}
```

```
public class A {
  private int i = 0;
  private int foo() {
    i = i + 1;
    return i;
  }
  public A() {
    if (foo() == 0 || foo() != 0) {
      System.out.println(i);
    }
  }
}
```

```
public class A {
  private int i = 0;
  private int foo() {
    return i++;
  }
  public A() {
    if (foo() == 0 || foo() != 0) {
      System.out.println(i);
    }
  }
}
```

1	2	1

I det første tilfellet gir det første kallet til **foo() 1** og det første leddet i **if**-testen dermed **true**. I det andre tilfellet gir det første leddet **false** og **foo()** utføres en ekstra gang. I det tredje tilfellet returnerer det første kallet til **foo() 0**, selv om/samtidig som **i** økes og det første leddet i **if**-testen gir **true**.

d) Anta følgende metode-deklarasjon:

```
int foo(int i) {
  if (i > 0) {
    System.out.println("i: " + foo(i - 1));
  }
  return i;
}
```

Hva blir skrevet ut dersom en utfører **foo(2)**? Forklar hvorfor!

i: 0
i: 1
Kallet til foo(i-1) utføres før kallet til println (selve utskriften). Derfor vil utskriften fra kallet til foo(1) komme før utskriften fra kallet til foo(2).