Institutionen för
Datavetenskap
CTH, GU

HT08
TDA550, DIT720
08-12-09

T E N T A M E N för

Objektorienterad programvarutveckling IT, fk TDA550, DIT720

DAG : 15 december 2008 Tid : 8.30-12.30 SAL : H

Ansvarig	:	Bror Bjerner, tel 772 10 29, 55 54 40
Resultat	:	senast 9/1 -09
Hjälpmedel	:	X.Jia: *Object-Oriented Software Development Using Java edition 1 och 2*, eller kopierat utdrag
Betygsgränser	:	**CTH** 3 : 24 p, 4 : 36 p, 5 : 48 p
Betygsgränser	:	**GU** Godkänt : 24 p, Väl godkänt : 45 p

OBSERVERA NEDANSTÅENDE PUNKTER

- Börja varje ny uppgift på nytt blad.
- Skriv din tentakod på varje blad.
- Använd bara ena sidan på varje blad.
- Klumpiga, komplicerade och/eller oläsliga delar kan ge poängavdrag.
- **L y c k a t i l l ! ! !**

Uppg 1: Allmänna frågor. Svara med ja eller nej. **Obs:** Rätt svar ger 2 poäng, felaktigt svar ger **-1** poäng. Poängen för hela uppgiften kan däremot inte bli negativ.

a) Är det så att gränssnittet `java.util.SortedSet<E>` implementerar `java.util.Collection<E>` ?

(2 p)

b) Kan olika objekt, som är instanser av samma klass, komma åt varandras `private`-deklarerade variabler ? **(2 p)**

c) Antag att klass `A` är en subklass av klass `B`. Antag vidare att `B` endast har en konstruktor `B()`. Kan konstruktorn `B()` skrivas över av en konstruktor i klass `A` ? **(2 p)**

d) Antag att `A`, `B` och `C` är olika klasser. Kan typ `A` vara en subtyp av både typ `B` och `C`. **(2 p)**

e) Antag att `x.hashCode()` och `y.hashCode()` returnerar exakt samma värde. Kommer i så fall `x.equals(y)` alltid att returnera `true` ? **(2 p)**

Uppg 2: Gränssnittet `Comparator<E>` ser ut på följande sätt:

```
public interface Comparator<T> {
    int compare( T t1, T t2 );
}
```

Med instruktionen att `compare` skall returnera ett negativt heltal, noll eller ett positivt heltal, om det första argumentet är mindre än, lika med respektive större än det andra argumentet.

(Jag har här medvetet utelämnat equals).

a) Deklarera en klass `StringLengthComparator` som implementerar en `Comparator` för strängar, enligt:

De två strängarna skall i första hand jämföras med avseende på längd, där en kortare sträng alltid är 'mindre' än en längre sträng. Endast om strängarna är lika långa skall de jämföras lexikografiskt.

(Dvs enligt den redan implementerade `compareTo`-metoden för strängar).

(5 p)

b) Deklarera vidare en klass `ReverseStringComparator` som också implementerar en `Comparator` för strängar. Du skall nu jämföra de *omvända strängarna* lexikografiskt.

(Dvs att t.ex. "cba" skall vara mindre än "aab", eftersom "abc" är mindre än "baa" enligt den vanliga ordningen).

Tips: För att vända en sträng `s` kan du använda:

```
(new StringBuffer(s).reverse()).toString()
```

(3 p)

c) Slutligen skall du skriva ett program, som använder de två komparatorerna. Programmet skall sortera och skriva ut argumenten på kommandoraden två gånger. Första gången enligt sorteringsordningen given av
`StringLengthComparator` och andra gången enligt sorteringsordningen given av
`ReverseStringComparator`

Tips: För sortering, använd `Collections` metod:

```
public static <T> void
   sort( List<T> list, Collator<? super T> c )
```

som sorterar den lista som ges som argument.

(4 p)

Uppg 3: Uppgiften går ut på att skriva ett program `MultiReplace`, som ersätter flera ord i en textfil med andra ord och skriver resultatet i en annan given fil.

Exempel: Antag att vi har följande indatafil `kalle.txt`:

```
Kalle bor i Ankeborg.
Kajsa bor också i Ankeborg.
```

Låt oss nu exekvera:

```
> java MultiReplace kalle.txt kajsa.txt Kalle
jag Kajsa du Ankeborg Göteborg
```

Då skall filen `kajsa.txt` efteråt innehålla:

```
jag bor i Göteborg
du bor också i Göteborg
```

Som kan ses av exemplet, så tar programmet ett jämnt antal argument, först in- och ut-datafil följt av en serie av par av ord. (Ordet som skall ersättas och vad det skall ersättas med).

För att förenkla uppgiften, behöver du inte bekymra dig om stora och små bokstäver och vidare skall bara hela ord bytas ut.

Du får inte implementera hela uppgiften i en klass, utan du skall göra en klass för var och en av följande deluppgifter:

a) En klass `Replacer`. Denna klass ansvarar för själva ersättandet av orden, inkluderat läsandet och skrivandet från/på filerna. Däremot är den inte ansvarig för hanterandet av tolkandet av argumenten på kommandoraden. När `Replacer` skapas, så skall den via konstuktorn bli given en `Map`, som innehåller orden som skall ersättas och vilka ord de skall ersättas med.

Vidare skall det finnas en metod `replace`, som givet två strängar som argument med namnen på in- och ut-datafilen, utför **hela** det specifierade arbetet.

Uppstår det något `IOEexception` skall det bara kastas vidare, och tas hand om i den andra klassen.

(8 p)

b) Den andra klassen är själva `MultiReplace`. I denna klass tar du hand om argumenten på kommandraden och hanterar eventuella fel som kan uppstå. Om `IOEexception` upstår vid anrop av `replace`, skall det ges en felutskrift : An IO problem has occured

(6 p)

Uppg 4: Collections Framework innehåller ett flertal *abstrakta* klasser som faktoriserar ut gemensam kod från konkreta samlingsklasser och som fungerar som en bas för nya implementeringar. T.ex. i ramverket har vi `AbstractCollection<E>`, som är en abstrakt klass som implementerar `Collection<E>`.

För att göra det enklare skall vi i stället använda gränssnittet `ExamCollection`:

```
import java.util.*;
public interface ExamCollection<E> {
    boolean     add ( E e );
    void        clear();
    boolean     contains( Object o );
    boolean     isEmpty();
    Iterator<E> iterator();
    boolean     remove( Object o );
    int         size();
}
```

Detta gränsssnitt har inga valbara metoder, dvs alla metoder måste implemeneteras.

Skriv en abstrakt klass som implementerar gränssnittet `ExamCollection` och som har tre abstrakta metoder: `add`, `iterator` och `size`. De övriga 4 metoderna måste vara konkreta, dvs inte abstrakta. (Du hittar API-definitionen för dessa metoder samt API:n för iteratorer längst bak i häftet).

Din klass `AbstractExamCollection` får **inte** ha några instansvariabler.

Kom ihåg att generella samlingar kan tillåta eller inte tillåta `null`-referenser, dvs motsvarande gäller då för argumenten till `add`, `contains` och `remove`. Vi delar därför upp problemet i två delproblem:

a) I den första versionen av `AbstractExamCollection` kan du anta att `contains` och `remove` aldrig anropas med `null` som argument. **(10 p)**

b) Nu skriver du om metoderna `contains` och `remove` så att de även fungerar korrekt då argumentet tillåts vara `null`. **(4 p)**

Uppg 5: Givet följande program `Person.java`:

```
public class Person extends Thread {

 private String name;

 private Person whomToAsk;

 public Person( String name ) {
    this.name = name;
 }

 public void setWhomToAsk( Person whomToAsk ) {
    this.whomToAsk = whomToAsk;
 }

 public String toString() {
    return name;
 }

 private void getAQuestionFrom( Person p ) {
   try {
     System.out.println(
      this + " says: I got a question from " + p );
       // Tänka en stund före svaret
     Thread.sleep(50);
     p.getAnAnswerFrom(this);
   } catch( InterruptedException e) {}
 }

 private void getAnAnswerFrom( Person p ) {
    System.out.println(
      this + " says: I got an answer from " + p );
 }

 public void run() {
    whomToAsk.getAQuestionFrom(this);
 }
}
```

Dina uppgifter är nu:

a) Svara med egna ord vad varje metod gör. **(2 p)**

b) Definiera en testklass `QuestionsAndAnswers`, som i sin `main`-metod kostruerar två personer Bror och Michael av typen `Person` som frågar varandra, dvs Bror får en fråga från Michael och Michael får en fråga från Bror.

Notera: Klassen `QuestionsAndAnswers` kan inte direkt anropa alla metoderna i `Person` eftersom de är `private`. Istället måste `main` starta trådar för varje person.

(3 p)

c) Ge ett möjligt exempel på hur utskriften kan se ut när din `main`-metod i `QuestionsAndAnswers` exekveras. **(1 p)**

d) Antag nu att vi nu vill införa ett begränsning: En `Person` får inte störas av någon medan han 'tänker'. En `Person` skall t.ex. inte kunna få en `getAnAnswerFrom` från någon, medan han ´tänker', dvs så länge `Thread.sleep(50)`.

För att åstadkomma detta kan vi använda synkronisering. Låt oss därför i stället definiera metoderna som:

```
private synchronized void getAQuestionFrom(..) {
   ...
}

private synchronized void getAnAnswerFrom(..) {
   ...
}
```

Om vi nu exekverar `QuestionsAndAnswers`, kan det hända att vare sig Bror eller Michael får något svar, dvs programmet terminerar inte. Förklara varför och ge en analys om när detta kan hända. (Enklast genom att beskriva ett exempel).

(4 p)