Institutionen för
Datavetenskap
CTH, GU

HT08
TDA550, DIT720
09-04-07

T E N T A M E N för

Objektorienterad programvarutveckling IT, fk TDA550, DIT720

DAG : 14 april 2009 Tid : 8.30-12.30 SAL : V

Ansvarig	:	Bror Bjerner, tel 772 10 29, 55 54 40
Resultat	:	senast 27/4 -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 ger 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) Kommer kompilatorn att acceptera:

```
Collection<String> col
     = new TreeMap<String,String>();
```

om `col` inte är deklarerad tidigare ? **(2 p)**

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

c) Om A är en subklass till B, gäller då att `ArrayList<A>` är en subklass till `ArrayList<B>` ? **(2 p)**

d) Kan en 'abstract' klass ha instansvariabler ? **(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 `Comparable<T>` ser ut på följande sätt:

```
public interface Comparable<T> {
    int compareTo( T t );
}
```

Med instruktionen att `compareTo` skall returnera ett negativt heltal, noll eller ett positivt heltal, om objektet själv är mindre än, lika med respektive större än argumentet.

a) Deklarera en klass `MyString`, som

- endast innehåller en sträng, som ges via den enda konstruktorn,
- går att skriva ut,
- implementerar `equals` fullständigt och
- implementerar `Comparable` fullständigt enligt:

  De två ingående 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).

(10 p)

b) Du skall nu skriva skriva ett program, som skall sortera och skriva ut argumenten på kommandoraden två gånger.

- Första gången enligt sorteringsordningen given av `MyString`,
- andra gången enligt naturliga sorteringsordningen för strängar och
- eventuella dubletter skall bara skrivas ut en gång.

Tips: Använd någon samling (`Collection`) !

(5 p)

Uppg 3: Skriv ett program `FromPirate` som tar en fil på rövarspråket, med suffixet `.pirate`, på kommandoraden och översätter den till vanlig text. Resultatet skall skrivas ut på en textfil med samma namn som indatafilen, men med suffixet `.txt` i stället för `.pirate`.

Vid felaktigt eller inget argument på kommandoraden vid anrop skall ett felmeddelande ges, som visar hur programmet skall användas.

Att översätta till rövarspråket fungerar på följande sätt:

- Varje konsonant dubbleras och ett `o` sätts mellan konsonaterna,
- Om konsonaten är en stor bokstav, så blir dubbletten en liten bokstav.
- Vokaler och övriga symboler översätts rakt av, dvs blir samma symbol.

Om det visar sig att indata-filen ej består av rövarspråket kastas ett lämpligt exception, med beskrivande felmeddelande.

Om en fil `hej.pirate` innehåller

```
Hohejoj     Bobrororor.
Hohejoj dodå  Bobrororor.
```

skall anropet

```
> java FromPirate hej.pirate
```

skapa en fil `hej.txt` som innehåller:

```
Hej Bror.
Hej då Bror.
```

Notera att du måste följa radstrukturen, men antalet blanktecken mellan orden behöver bara var ett, oberoende av hur många det fanns i indata-filen. Vidare är det givetvis tillåtet att använda `Scanner` som finns i `util`-paketet. (Se bilaga)

(14 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`:

```
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();
    Object[]    toArray();
}
```

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 5 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) Deklarera 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)

b) 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)

Institutionen för
Datavetenskap
CTH, GU

VT09
TDA550, DIT720
09-04-14

Lösningsförslag till tentamen i Objektorienterad programvarutveckling IT, fk.

DAG : 14 april 2009

Uppg 1: **a)** Nej, `Map` är ingen `Collection`.

b) Ja, `private` är klassorienterat, inte objektorienterat.

c) Nej, generiska typer är bara subklasser till '?', dvs `ArrayList<A>` och `ArrayList<B>` är båda subklasser till `ArrayList<?>`

d) Ja, en 'abstract' klass kan ha både färdiga metoder och både instans- och klass-variabler !

e) Nej, normalt skall det vara tvärtom !

Uppg 2: **a)** Notera att vi måste implementera metoden `hashCode` för att vi fullständigt skall implementera `equals` !

```
public class MyString
       implements Comparable<MyString> {

 private String myString;

 public MyString( String theString ) {
   myString = theString;
 }

 public String toString() {
   return myString;
 }  // toString

 public boolean equals( Object o ) {
   if ( o instanceof MyString )
     return myString.equals( ((MyString)o).myString );
   else
     return false;
 } // equals

 public int hashCode() {
   return myString.hashCode();
 } // hashCode

 public int compareTo( MyString compString ) {
   int jfr = myString.length() -
             compString.myString.length();
   if ( jfr == 0 )
     return myString.compareTo(
                         compString.myString );
   else
     return jfr;
 } // compareTo

} // MyString
```

b) Genom att välja sorterade mängder, så löser vi alla våra problem !

```
import java.util.*;

public class SortArgs {

 public static void main( String[] args ) {

  SortedSet<MyString> setOfMyString =
                        new TreeSet<MyString>();
  SortedSet<String>   setOfString =
                        new TreeSet<String>();

  for( String s : args ) {
    setOfMyString.add( new MyString( s ) );
    setOfString.add( s );
  }

  System.out.println( setOfMyString );
  System.out.println( setOfString );

 } // main

} //  SortArgs


/* En liten testkörning
java SortArgs kalle kajsa Tjatte Knatte Fnatte kalle
[kajsa, kalle, Fnatte, Knatte, Tjatte]
[Fnatte, Knatte, Tjatte, kajsa, kalle]
 */
```

Uppg 3:

```
import java.util.*;
import java.io.*;

public class FromPirate {

  private static String small
          = "bcdfghjklmnpqrstvwxz";
  private static String noPirate
          = "Argument file is a Non-pirate file";

  private static String translateWord( String s ) {

     int i    = 0,
         max  = s.length();
     String res = "";
     while ( i < max ) {
       char c = s.charAt(i);
       res = res + c;
       int index  = small.indexOf(Character.toLowerCase(c));
       if (index > -1 )
         try {
           if ( s.charAt( i + 1 ) == 'o' &&
                s.charAt( i + 2 ) == small.charAt( index ) )
             i = i + 2;
           else
             throw new IllegalArgumentException(noPirate);
         }
         catch(IndexOutOfBoundsException ioobe) {
             throw new IllegalArgumentException(noPirate);
         }
       i++;
     }
     return res;
  }
```

```
  public static void main( String[] args ) {
    if (args.length != 1 || ! args[0].endsWith(".pirate")){
      System.err.println(
               " Usage: java FromPirate <file name>.pirate" );
      System.exit(0);
    }

    String outFile =
      args[0].substring(0,args[0].length()- 6)+"txt";
    Scanner     pirate = null;
    PrintWriter out    = null;
    try {
      pirate = new Scanner( new File(args[0] ));
      out = new PrintWriter(
                new BufferedWriter(
                    new FileWriter( outFile )));
    }
    catch ( FileNotFoundException fnfe ) {
      System.err.println(" Could not find file: "+ args[0]);
      System.exit(0);
    }
    catch ( IOException fnfe ) {
       System.err.println( "Something rotten in Denmark" );
       System.exit(0);
    }
    try {
      while ( pirate.hasNextLine() ) {
        Scanner line = new Scanner( pirate.nextLine() );
        while ( line.hasNext() )
          out.print( translateWord(line.next()) + " " );
        out.println();
        line.close();
      }
    }
    finally {
      pirate.close();
      out.close();
    }
  } // main
} // FromPirate
```

Uppg 4: **a)**

```
import java.util.*;

public abstract class AbstractExamCollection<E>
                implements ExamCollection<E>{

    public abstract boolean add ( E e );

    public void  clear() {
        Iterator<E> it = iterator();
        while ( it.hasNext() ) {
            it.next();
            it.remove();
        }
    }

    public boolean contains( Object o ) {
       Iterator<E> it = iterator();
       while ( it.hasNext() )
          if ( o.equals( it.next() ))
             return true;
       return false;
    }

    public boolean isEmpty() {
       return size() == 0;
    }

    public abstract Iterator<E> iterator();

    public boolean remove( Object o ) {
       Iterator<E> it = iterator();
       while ( it.hasNext() )
          if ( o.equals( it.next() )) {
            it.remove();
            return true;
          }
       return false;
    }
```

```
    public abstract int size();

    public Object[] toArray() {
       Object[] res = new Object[size()];
       Iterator<E> it = iterator();
       for( int i = 0; i < res.length; i++ )
           res[i] = it.next();
       return res;
    }
}
```

b)
```
public boolean contains( Object o ) {
  Iterator<E> it = iterator();
  if ( o == null )
     while ( it.hasNext() )
       if ( it.next() == null )
          return true;
  else
     while ( it.hasNext() )
       if ( o.equals( it.next() ))
          return true;
  return false;
}

public boolean remove( Object o ) {
  Iterator<E> it = iterator();
  if ( o == null )
    while ( it.hasNext() )
      if ( it.next() == null ) {
        it.remove();
        return true;
      }
  else
    while ( it.hasNext() )
      if ( o.equals( it.next() )) {
         it.remove();
         return true;
      }
  return false;
}
```

Uppg 5: **a)**

```
public class QuestionsAndAnswers {

  public static void main( String[] args ) {
    Person bror    = new Person("Bror");
    Person michael = new Person("Michael");

    bror.setWhomToAsk(michael);
    michael.setWhomToAsk(bror);

    michael.start();
    bror.start();
  }
}
```

b)

```
Michael says: I got a question from Bror
Bror says: I got a question from Michael
```

Efter detta stannar programmet eftersom vi har en låsning, Bror väntar på Michael och Michael väntar på Bror.