Institutionen för
Datavetenskap
CTH, GU

HT07
TDA550, DIT720
08-03-18

T E N T A M E N för

Objektorienterad programvarutveckling IT, fk TDA550, DIT720

DAG : 25 mars 2008 Tid : 14.00-18.00 SAL : V

Ansvarig	:	Bror Bjerner, tel 772 10 29, 55 54 40
Resultat	:	senast 7/4 -08
Hjälpmedel	:	X.Jia: *Object-Oriented Software Development Using Java edition 1 och 2*
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 personnummer 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) Kan en 'abstract' klass ha instansvariabler ? **(2 p)**

b) Kan ett 'interface' ha variabler ? **(2 p)**

c) Givet:

```
List<Integer>[] ali = new List<Integer>[10];
for ( int i = 0; i < 10; i++ )
     ali[i] = new ArrayList<Integer>();
```

Är typen för `ali[5]` lika med `ArrayList<Integer>` ?
(2 p)

d) Kommer kompilatorn att acceptera:

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

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

e) Kommer kompilatorn att acceptera:

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

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

Uppg 2: Skriv ett program `ToPirate` som tar en textfil med suffixet `.txt` på kommandoraden och översätter den till rövarapråket. Resultatet skall skrivas ut på en textfil med samma namn som indatafilen men med suffixet `.pirate` i stället för `.txt`.

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

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 en fil `hej.txt` innehåller

```
Hej Bror.
Hej då Bror.
```

skall anropet `> java ToPirate hej.txt` skapa en fil `hej.pirate` som innehåller:

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

(18 p)

Uppg 3: Givet följande gränssnitt:

```
public interface Stack<E> {
    /**
     * Lägg till argumentet överst i stacken.
     * Om argumentet är null
     * kastas NullPointerException.
     */
    public void push( E elem );

    /**
     * Tag bort det översta  elementet i stacken
     * och returnera det.
     * Kasta NoSuchElementException
     * om stacken är tom.
     */
    public E pop();

    /**
     * Returnera översta elementet i stacken.
     * Kasta NoSuchElementException
     * om stacken är tom.
     */
    public E top();

    /**
     * Returnera antalet element i stacken.
     */
    public int size();

} // inteface Stack
```

En `Stack<E>` är en datastruktur där nya element alltid läggs överst och där det alltid är det översta elementet som hämtas. Dvs. den följer LIFO-principen (Last In First Out).

Din uppgift är att definiera en klass `MyStack<E>` som implementerar gränssnittet `Stack<E>`. Implementeringen **skall** delegera det mesta av sitt jobb till datastrukturen `LinkedList<E>` i `util`-paketet. (se utdrag i bilaga). **(12 p)**

Uppg 4: Säg först att vi har en namntyp:

```
public class Namn {
    public final String efternamn;
    public final String[] förnamn;
    // kostruktorer och metoder
} //  Namn
```

Vidare givet att vi har en mängd personer, med den enkla implementeringen

```
public class Person {
   protected String idNummer;
   protected Namn namn;

   public Person( String idNummer, Namn namn ) {
       this.idNummer = idNummer;
       this.namn     = namn;
   }

   public String getIdnummer() {
      return idNummer;
   }
   public Namn getNamn() {
      return name;
   }
}
```

Vidare har vi samlat ihop en massa personer i ett en mängd `Set<Person>`, som vi nu vill sortera efter namnet i lexikografisk ordning (dvs enligt ASCII-kodens ordning). I första hand efter efternamnet och därefter förnamnen i den ordning de förkommer i fältet.

För att sortera dem, så skall du använda dig av `SortedSet<E>`. Problemet är bara att `Person` inte är jämförbar. Du skall därför göra en subklass `JfrPerson` till `Person` som är jämförbar enligt gränssnittet `Comparable<E>` och där 'minsta' `JfrPerson` är den person som kommer först i lexikografisk ordning

Vidare skall i denna subklass finnas en klassmetod som tar ett `Set<Person>` som argument och som resultat ger en textsträng, där det finns en person per rad i lexikografisk ordning.

Slutligen vill vi även göra en riktig `equal`-metod, vilket i sin tur medför att vi även måste göra en ny `hashCode`-metod. Eftersom id-numren skall vara unika, så räcker det att använda dessa idnummer för dessa metoder.
Poängfördelning:

a) Klasshuvud och konstruktor **(4 p)**

b) `compareTo`-metoden **(5 p)**

c) `toString`-metoden **(5 p)**

d) `equal`-metoden **(3 p)**

e) `hashCode`-metoden **(3 p)**

Institutionen för
Datavetenskap
CTH, GU

VT07
TDA550, DIT720
08-13-19

Lösningsförslag till tentamen i Objektorienterad programvarutveckling IT, fk.

DAG : 25 mars 2008

Uppg 1:

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

b) Ja, men endast klass-konstanter, dvs definierade med `static final`

c) Nej, typen för `ali[5]` är `List<Integer>`

d) Nej, `Map` är ingen `Collection`.

e) Ja, `Map` är ett `Object`.

Uppg 2:

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

public class ToPirate {

  public static void main(String[] args) {
    String useString = "Usage: java ToPirate <file>.txt";
    String cons      = "bcdfghjklmnpqrstvxBCDFGHJKLMNPQRSTVX";
    Scanner in       = null;
    PrintWriter out  = null;
    try {
      if ( !  args[0].endsWith(".txt" ) ) {
        System.err.println( useString ); System.exit(0);
      }
      in  = new Scanner( new File( args[0] ) );
      out = new PrintWriter(
              new BufferedWriter (
                new FileWriter(
                  args[0].substring(0,args[0].length()-3)
                  + "pirate" )));
    }
    catch (IndexOutOfBoundsException  ie ) {
      System.err.println( useString ); System.exit(0);
    }
    catch( IOException fe ) {
      System.err.println("Filen " +  args[0] + " finns ej" );
      System.exit(0);
    }
    while ( in.hasNextLine() ) {
      String s = in.nextLine();
      for ( char c : s.toCharArray() )
        if ( cons.indexOf(c) > -1 )
          out.print( c + "o" + Character.toLowerCase(c) );
        else
          out.print( c );
      out.println();
    }
    in.close();
    out.close();
  }
}
```

Uppg 3:

```
import java.util.*;

public class MyStack<E> implements Stack<E> {

    protected LinkedList<E> stack = new  LinkedList<E>();

    public void push( E elem ) {
        if ( elem == null )
           throw new NullPointerException(
                 "Null element not accepted in stack" );
        else
           stack.addFirst( elem );
    }

    public E pop() {
        if ( stack.isEmpty() )
           throw new NoSuchElementException(
                                "Empty stack in pop" );
        else
           return  stack.removeFirst();
    }

    public E top() {
        if ( stack.isEmpty() )
           throw new NoSuchElementException(
                                "Empty stack in top" );
        else
           return  stack.getFirst();
    }

    public int size() {
        return stack.size();
    }

} // MyStack
```

Uppg 4:

```
import java.util.*;

public class JfrPerson
       extends Person
       implements Comparable<JfrPerson> {

  public JfrPerson( String idNummer,
                    Namn   namn       ) {
    super( idNummer, namn );
  } // constructor PersonByAge

  public int compareTo( JfrPerson jp ) {
    int i        = 0;
    int maxIndex = Math.min( namn.förnamn.length,
                             jp.namn.förnamn.length );
    int jfr = namn.efternamn.compareTo( jp.namn.efternamn );
    while ( jfr == 0 && i < maxIndex ) {
      jfr = namn.förnamn[i].compareTo( jp.namn.förnamn[i] );
      i = i + 1;
    }
    if ( jfr == 0 )
      return namn.förnamn.length - jp.namn.förnamn.length;
    else
      return jfr;
  } // compareTo

  public static String toString( Set<Person> sp ) {
    SortedSet<JfrPerson> ssp = new TreeSet<JfrPerson>();
    String res = "";
    for ( Person p : sp )
      ssp.add( new JfrPerson( p.getIdNummer(), p.getNamn() ));
    for (JfrPerson jp : ssp ) {
      Namn namn = jp.getNamn();
      res = res + jp.getIdNummer() + " " + namn.efternamn;
      for (String s : namn.förnamn )
        res += " " + s;
      res += "\n";
    }
    return res;
  } // toString
```

```
  public boolean equals( Object o ) {
    if ( o instanceof JfrPerson )
      return idNummer.equals( ((JfrPerson) o).idNummer );
    else
      throw new ClassCastException();
  } // equals

  public int hashCode() {
    return idNummer.hashCode();
  } // equals

} // class JfrPerson
```