Institutionen för
Datavetenskap
CTH, GU

VT08
TDA550, DIT720
09-12-12

Lösningsförslag till tentamen i Objektorienterad programvarutveckling IT, fk.

DAG : 15 december 2008

Uppg 1: **a)** Nej, ett gränssnitt implementerar ingenting.

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

c) Nej, konstruktorer ärvs över huvud taget inte, och kan därför inte skrivas över.

d) Ja, om t.ex. `A` är subklass till `B` och `B` är subklass till `C`

e) Nej, normalt skall det vara tvärtom !

Uppg 2: **a)**

```
import java.util.*;

public class StringLengthComparator
       implements Comparator<String>       {

 public int compare( String s1, String s2 ) {
   int diff = s1.length() - s2.length();
   if ( diff != 0 )
     return diff;
   else
     return s1.compareTo( s2 );
 }
}
```

b)

```
import java.util.*;

public class ReverseStringComparator
       implements Comparator<String> {

 public int compare( String s1, String s2 ) {
  String rs1 =
   (new StringBuffer(s1).reverse()).toString();
  String rs2 =
   (new StringBuffer(s2).reverse()).toString();

  return rs1.compareTo( rs2 );
 }
}
```

c)

```
import java.util.*;

public class SortStringTest {

 public static void main( String[] args ) {

  List<String> argLi = Arrays.asList( args );

  Collections.sort(
       argLi, new StringLengthComparator() );
    System.out.println();
    System.out.println( argLi );

  Collections.sort(
      argLi, new ReverseStringComparator() );
  System.out.println();
  System.out.println( argLi );
 }
}
```

Uppg 3: **a)**

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

public class Replacer {

 private Map<String,String> replaceMap;

 public Replacer( Map<String,String> map ) {
     replaceMap = map;
 }

 public void replace( String inFile, String outFile )
             throws IOException                      {
   BufferedReader in =
       new BufferedReader( new FileReader( inFile ));
   PrintWriter   out =
       new PrintWriter(new FileWriter( outFile ));
   String row = in.readLine();
   while ( row != null ) {
     Scanner sc = new Scanner( row );
     if ( sc.hasNext() ) {
       String word = sc.next();
       String replaceWord = replaceMap.get( word );
       if ( replaceWord == null ) out.print( word );
       else                out.print( replaceWord );
     }
     while ( sc.hasNext() ) {
       out.print(" ");
       String word = sc.next();
       String replaceWord = replaceMap.get( word );
       if ( replaceWord == null )
 out.print( word );
       else
 out.print( replaceWord );
     }
     out.println();
     row = in.readLine();
   }
   in.close(); out.close();
 }
}
```

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

public class MultiReplace {

 public static void main( String[] args ) {

   String           inputFile  = null,
                    outputFile = null;
   Map<String,String> replaceMap =
              new HashMap<String,String>();

   try {
     inputFile  = args[0];
     outputFile = args[1];
     for( int i = 2; i < args.length; i = i+2 )
       replaceMap.put( args[i], args[i+1] );
   }
   catch (IndexOutOfBoundsException ioobe) {
     System.err.println("Wrong number of arguments");
   }
   try {
     new Replacer( replaceMap ).replace( inputFile,
                                         outputFile );
   }
   catch (IOException ioe) {
     System.err.println("An IO problem has occured");
   }
 }
}
```

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();

}
```

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)** Ges ej här.

b)
```
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();
  }
}
```

c)
```
Bror says: I got a question from Michael
Michael says: I got a question from Bror
Michael says: I got an answer from Bror
Bror says: I got an answer from Michael
```

d)
```
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.