

Inhaltsverzeichnis

1 com.strato.framework	1
1.1 CAlphabet.....	1
1.2 CRule.....	7
1.3 CString	10
1.4 CStringStreamable	16
1.5 ENoSuchCharacterException.....	19
1.6 IAccess	20
1.7 IInput	21
1.8 IOutput	22
1.9 IPersistence.....	23
1.10 IStreamable.....	24
1.11 IString.....	25
2 com.strato.framework.run.....	26
2.1 CAbstractCase.....	26
2.2 CRunner	27
2.3 CSTSICase	30
3 com.strato.framework.test	31
3.1 CAlgorithmDescriptor.....	31
3.2 CConvention.....	33
3.3 CCtrlCenter	36
3.4 CExactMatchingTestInfo	38
3.5 CGenericDescriptor.....	40
3.6 CParameterDescriptor	42
3.7 CReferenceDescriptor	45
3.8 CSimpleTestInfo	46
3.9 CTestDescriptor	49
3.10 EConventionException.....	51
3.11 EInvalidUserInputException.....	52
3.12 EUpdateException.....	53
3.13 IGraphicResult	54
3.14 IResult	55
3.15 ITestInfo	56
3.16 ITextResult	57
4 com.strato.lib	58
4.1 CRandom.....	58
5 com.strato.lib.alphabet	61
5.1 CLowerCase	61

5.2	CNotQuiteASCII.....	62
5.3	CProteine.....	63
5.4	CUserDefined.....	64
6	com.strato.lib.ds	66
6.1	CSuffixTree.....	66
7	com.strato.lib.output	69
7.1	CBarPlotAssistant	69
7.2	CLinePlotAssistant.....	70
7.3	CPlotAssistant	71
8	com.strato.lib.rule	73
8.1	CPalindromOfLength	73
8.2	CPatternAtLeastNTimes	74
9	com.strato.tests	76
9.1	AllSubmassesFindingProblem	76
9.2	BoyerMoore	77
9.3	MultiplicityVectors	80
9.4	OneStringMassFindingProblem	83
9.5	OSMFPBinsearch.....	85
9.6	OSMFPLinsearch	88
9.7	Statistics	90
9.8	Submasses	93
9.9	Substrings	95
9.10	SuffixTreeSearch.....	98
9.11	WSIntervalWeight.....	101
9.12	WSSubmasses	104

1 com.strato.framework

1.1 CAlphabet

```

package com.strato.framework;

/*
 * @(#)CAlphabet.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import java.util.*;

/**
 * This class represents an abstract superclass of all alphabets.
 * Because of the singleton behavior of each Alphabet, use the
 * <code>getInstanceByName</code> method instead of <code>new</code> to
 * create a new instance of any available subclass of CAlphabet.
 * <p />
 * The main features of a the alphabet class is the character set of UNICODE
 * characteres, which forms the alphabet. To each character itself a weight may
 * be assigned. Standard weight is <code>1.0</code>. Finally a descprition
 * string e.g. the biological name of a proteine alphabet may be associated.
 * <p />
 * Each Alphabet has a JOKER character at index 0, which represents a not defined
 * character (place holder, ambiguous letter, wildcard) and will be used to
 * fill a newly created instance of {@link CString} with a given length.
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.CString
 * @see com.strato.lib.alphabet.CAlphaNumeric
 * @see com.strato.lib.alphabet.CProteine
 * @see com.strato.lib.alphabet.CUserDefined
 * @since STRATO 1.0
 */

public abstract class CAlphabet implements IString, java.io.Serializable {

    /** The symbol representing of an undefined character */
    public static final char JOKER = '$';

    /** The index always used to refer the JOKER Symbol */
    public static final byte JOKER_INDEX = 0;

    /** An alphabet the consists of the characters occuring in a given string */
    public static final int USERDEFINED_ALPHABET = 0;

    /** An alphabet that consists of quite alle US-ASCII Chars */
    public static final int DEFAULT_ALPHABET = 1;

    /** Alphabet of all latin lowercase letters */
    public static final int LC_ALPHABET = 2;

    /** Alphabet of all latin uppercase letters */
    public static final int UC_ALPHABET = 3;

    /** Alphabet of all arabian digits */
    public static final int DIGITS_ALPHABET = 4;

    /** Alphabet formed by the union of LC, UC, DIGIT */
    public static final int ALPHANUM_ALPHABET = 5;

    /** Alphabet of lowercase greek letters */
    public static final int GREEK_ALPHABET = 6;
}

```

```

/** Alphabet of two characters */
public static final int BINARY_ALPHABET = 7;

/** Alphabet of Deoxyribonucleic Acid Sequences */
public static final int DNA_ALPHABET = 8;

/** Alphabet consisting of the 20 commonly occurring Amino Acid Residues */
public static final int PROTEINES_ALPHABET = 9;

private static final String[] ALPHABETS = {
    "com.strato.lib.alphabet.CUserDefined",
    "com.strato.lib.alphabet.CNotQuiteASCII",
    "com.strato.lib.alphabet.CLowerCase",
    "com.strato.lib.alphabet.CUpperCase",
    "com.strato.lib.alphabet.CDigits",
    "com.strato.lib.alphabet.CAlphaNumeric",
    "com.strato.lib.alphabet.CHexadecimal",
    "com.strato.lib.alphabet.CBinary",
    "com.strato.lib.alphabet.CDna",
    "com.strato.lib.alphabet.CProtein"
};

/**
 * Returns the number of predefined alphabets listed as constants
 */
public static int getNumberOfPredefinedAlphabets() {
    return ALPHABETS.length;
}

/* flag to ensure that user never creates a new instance using the new operator */
private static boolean singletonException;
/* manages all alphabet singleton instances */
private static Map instanceMap;

/* static initializer */
static {
    singletonException = true;
    instanceMap = Collections.synchronizedMap(new HashMap());
    for(int i=0; i<ALPHABETS.length; i++) {
        getInstanceByName(i);
    }
}

/* for each Alphabet */

/** set this to a value != null to set other defaults than the standard */
protected double[] defaultWeights = null;
/** set this to a value != null to set other defaults than the standard */
protected String[] defaultDescriptions = null;

/** contains the current content of the alphabet */
protected char[] chars = null;
/** a list of the assigned weights to each character [default is 1.0] */
protected double[] weights = null;

/** the name to be displayed in any User Interface */
private String displayName = null;
/** a list of the assigned descriptions to each character [default is 1.0] */
private String[] descriptions = null;
/** a fast access structure to speed up the getIndexOf(...) method */
private Map keyMap = null;

/**
 * Creates an alphabet consisting of the specified charset
 *
 * @param displayName Name to be displayed in Choices like JComboBox
 * @param charset The content of the alphabet to be generated
 */
protected CAlphabet(String displayName, char[] charset) {
    if (singletonException) throw new RuntimeException("use CAlphabet.getInstanceByName(...)" +
instead of new");
    this.displayName = displayName;
    this.chars = charset;
    initProperties();
}

/**
 * Creates an alphabet having as charset the characters of a given string

```

```

/*
 * @param displayName Name to be displayed in Choices like JComboBox
 * @param charset     The content of the alphabet to be generated specified as string
 */
protected CAlphabet(String displayName, String charset) {
    this(displayName, charset.toCharArray());
}

/**
 * Initializes all the properties of an alphabet
 */
protected final void initProperties() {
    this.setWeights(null);          // default
    this.setDescriptions(null);     // default
    this.keyMap = createKeyMap();
}

// create an index for fast access //
private Map createKeyMap() {
    int size = this.getSize();
    Map keyMap = Collections.synchronizedMap(new HashMap(size));
    for(byte i=0; i<size; i++) {
        keyMap.put(new Character(chars[i]), new Byte(i));
    }
    return keyMap;
}

/**
 * call this method to get an instance of an alphabet by specifying its full
 * classname (including the package structure)
 * For example: className = com.strato.lib.alphabet.CProteine
 *
 * @return a reference to the desired alphabet (singleton) instance
 */
public static CAlphabet getInstanceByName(String className) {
    CAlphabet instance = (CAlphabet)instanceMap.get(className);
    if (instance == null) {
        try {
            CAlphabet.singletonException = false;
            instance = (CAlphabet)(Class.forName(className)).newInstance();
        }
        catch (Exception e) {
            instance = null;
        }
        finally {
            CAlphabet.singletonException = true;
        }
        instanceMap.put(className, instance);
    }
    return instance;
}

/**
 * Call this method to get an instance of an alphabet by specifying its classname
 * as one of the predefined constants in CAlphabet
 * For example: id = CAlphabet.LC_ALPHABET
 *
 * @return a reference to the desired alphabet (singleton) instance
 */
public static CAlphabet getInstanceByName(int id) {
    return getInstanceByName(ALPHABETS[id]);
}

/**
 * Returns the numbers of characters (including the mandatory JOKER character)
 *
 * @return number of the characters which form the alphabet
 */
public final int getSize() {
    return chars.length;
}

/**
 * Returns the index of a given character in the alphabet
 * @exception ENoSuchCharacterException if the <code>char</code>
 * argument is not a member of the assigned alphabet
 * @return index entry of a given character, -1 if character is not found
 */

```

```

public final byte getIndexOf(char ch) {
    Object iObj = keyMap.get(new Character(ch));
    if (iObj == null)
        throw new ENoSuchCharacterException("Character '" + ch + "' is not an element of this
alphabet");
    else return ((Byte)iObj).byteValue();
}

/**
 * Returns the character corresponding to a given index of the alphabet
 * @exception IndexOutOfBoundsException if the <code>index</code>
 * argument is negative or not less than the size of this
 * alphabet.
 * @return character at the specified index
 */
public final char getChar(int index) {
    if ((index < 0) || (index >= chars.length))
        throw new IndexOutOfBoundsException("index must be in [0," + (chars.length-1) + "]");
    return chars[index];
}

/**
 * Returns the weight of a character at a given index
 * @exception IndexOutOfBoundsException if the <code>index</code>
 * argument is negative or not less than the size of this
 * alphabet.
 * @return character weight of the specified index
 */
public final double getWeightOf(int index) {
    if ((index < 0) || (index >= chars.length))
        throw new IndexOutOfBoundsException("index must be in [0," + (chars.length-1) + "]");
    return weights[index];
}

/**
 * Set the weight of a character at a given index
 * @exception IndexOutOfBoundsException if the <code>index</code>
 * argument is negative or not less than the size of this
 * alphabet.
 * @param index index of the desired character to be modified
 * @param value the new weight to be assigned
 */
public final void setWeightOf(int index, double value) {
    if ((index < 0) || (index >= chars.length))
        throw new IndexOutOfBoundsException("index must be in [0," + (chars.length-1) + "]");
    weights[index] = value;
}

/**
 * Returns an array of weights of this alphabet
 *
 * @return array of character weights
 */
public final double[] getWeights() {
    return weights;
}

/**
 * Set the weights for all characters of an alphabet at once
 *
 * @param newValues an array of size (alphabet.getSize()) with weights
 */
public final void setWeights(double[] newValues) {
    if (newValues == null)
        weights = this.getDefaultWeights();
    else {
        System.arraycopy(newValues, 0, weights, 0, newValues.length);
    }
}

/**
 * Returns the default weights
 */
public final double[] getDefaultWeights() {

```

```

        double[] w = new double[chars.length];
        if (this.defaultWeights == null) {
            for(int i=0; i<chars.length; i++) w[i] = 1.0;
        }
        else {
            for(int i=0; i<chars.length; i++) w[i] = this.defaultWeights[i];
        }
        return w;
    }

    /**
     * Returns the description of a character at a given index
     * @exception IndexOutOfBoundsException if the <code>index</code>
     * argument is negative or not less than the size of this
     * alphabet.
     * @return description of a character at the specified index
     */
    public final String getDescriptionOf(int index) {
        if ((index < 0) || (index >= chars.length))
            throw new IndexOutOfBoundsException("index must be in [0,"+(chars.length-1)+"]");
        return descriptions[index];
    }

    /**
     * Set the character description for the character at a given index
     * @exception IndexOutOfBoundsException if the <code>index</code>
     * argument is negative or not less than the size of this
     * alphabet.
     * @param index index of the desired character to be modified
     * @param value the new description to be assigned
     */
    public final void setDescriptionOf(int index, String value) {
        if ((index < 0) || (index >= chars.length))
            throw new IndexOutOfBoundsException("index must be in [0,"+(chars.length-1)+"]");
        descriptions[index] = value;
    }

    /**
     * Set the description for characters of an alphabet at once
     *
     * @param newValues an array of size (alphabet.getSize()) with descriptions
     */
    public final void setDescriptions(String[] newValues) {
        if (newValues == null)
            descriptions = this.getDefaultDescriptions();
        else {
            System.arraycopy(newValues, 0, descriptions, 0, newValues.length);
        }
    }

    /**
     * Returns the default descriptions
     */
    public final String[] getDefaultDescriptions() {
        String[] d = new String[chars.length];
        if (this.defaultDescriptions == null) {
            d[0] = "JOKER";
            for(int i=1; i<chars.length; i++) d[i] = "<none>";
        }
        else {
            for(int i=1; i<chars.length; i++) d[i] = this.defaultDescriptions[i];
        }
        return d;
    }

    /**
     * Returns a String representing the alphabet, without weights and descriptions
     *
     * @return String representation of the Alphabet
     */
    public final String toString() {
        return displayName;
    }

    /* Implementation of interface IString */
    public String getAsString() {

```

```
    return new String(this.chars);
}

/* Implementation of interface IString */
public char[] getAsCharArray() {
    return this.chars;
}

/***
 * Main method (Only for testing purposes)
 */
public static void main(String[] args) {
    CAlphabet aGoodOne = CAlphabet.getInstanceByName(CAlphabet.DEFAULT_ALPHABET);
    // next line has to produce a RuntimeException
    CAlphabet aWrongOne = new com.strato.lib.alphabet.CAlphaNumeric();
}

} // eof CAlphabet
```

1.2 CRule

```

package com.strato.framework;

/*
 * @(#)CRule.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import java.util.*;

/**
 * This class forms the abstract superclass of all classes implementing any
 * sort of rule based string generation functionality.
 * <p />
 * The subclass have to implement the method createString(String params).
 * <code>params</code> represents the arguments delivered as {@link String}.
 * The arguments are separated by the comma character ',' and have the same
 * syntax as the Java Programming Language: e.g
 * <p /><blockquote><pre>
 * two enclosing " for class String:    -> "any String"
 * two enclosing ' for char           -> 'y'
 * suffix f for float:              -> 345.34f
 * suffix s for short:             -> -15s
 * suffix b for byte:              -> 12b
 * suffic l for long:              -> 5456456541
 * (true/false) for boolean:       -> true
 * suffix d or none for double:   -> -1.23
 * suffix i or none for int:      -> 129
 * </pre></blockquote>
 *
 * @author Daniel Emmenegger
 * @version 1.0 2001/03/03
 * @see com.strato.lib.rule.CPatternAtLeastNTimes
 * @see com.strato.lib.rule.CPalindromOfLength
 * @since STRATO 1.0
 */

public abstract class CRule {

    /** randomize instance */
    protected static Random random = new Random();

    /* the name to be displayed in GUIs */
    private String displayName = null;

    /* the descripton of the rule (viewed as tooltip) */
    private String descName = null;

    /** it contains the current parameters as String. */
    protected String params = null;

    /**
     * The standard constructor using this.getClass().getName() as name to be
     * displayed in any (Graphical) User Interface (GUI).
     */
    public CRule() {
        setDisplayName(this.getClass().getName());
    }

    /**
     * This constructor has to be called by all subclasses, if a name to be
     * displayed in GUI Controls is desired.
     */
    public CRule(String displayName) {
        setDisplayName(displayName);
    }

    /**

```

```

        * Set the currently displayed name
        *
        * @param displayName the name to be displayed
        */
    public void setDisplayName(String displayName) {
        this.displayName = displayName;
    }

    /**
     * Returns the description
     *
     * @return a description of the rule
     */
    public String getDescription() {
        return this.descName;
    }

    /**
     * Set the description
     *
     * @param desc the description to be displayed as tooltip
     */
    public void setDescription(String desc) {
        this.descName = desc;
    }

    /* inherited */
    public String toString() {
        return this.displayName;
    }

    /**
     * A template method to create a CString with the given parameters (in String
     * form) as input.
     * @exception RuntimeException if something goes wrong during creation of
     *          the string
     * @param params user arguments to this method the class represents
     * @param length the lenght of the CString to be generated
     * @param alphabet the alphabet the resulting CString depends on
     *
     * @return the created string by this class, or null if creation failed
     */
    public abstract CString createString(String params, int length, CAlphabet alphabet);

    /**
     * A very usefull method to extract parameters out of a specially formatted
     * String (the arguments are separated by commas, and they have Java Language
     * syntax), whereby they are returned as a Vector consisting of concret instances
     * with the specified content. Simple datatypes like int, boolean are wrapped.
     * @exception RuntimeException if something goes wrong during extraction of
     *          the string
     * @return a vector with the arguments as concrete instances
     */
    public static Vector extractParametersFromString(String args) throws RuntimeException {
        Vector argV = new Vector();
        if (!args.equals("")) { //has arguments

            StringTokenizer st = new StringTokenizer(args, ",");
            int count = 0;

            while(st.hasMoreTokens()) {
                try {
                    count++;
                    String token = st.nextToken();
                    String s = (new StringTokenizer(token)).nextToken();

                    if (s.indexOf("\"") != -1) { // isString
                        StringTokenizer t = new StringTokenizer(s, "\"", true);
                        while (t.hasMoreTokens()) {
                            s = t.nextToken();
                            if (s.charAt(0) == '\"') { // first "
                                s = t.nextToken();
                                if (s.charAt(0) == '\"') { // empty string
                                    argV.addElement("");
                                }
                            } else {
                                argV.addElement(new String(s));
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
        break;
    }
}
else if (s.indexOf("'''") != -1) { // is char
    StringTokenizer t = new StringTokenizer(s, "'''", true);
    while (t.hasMoreTokens()) {
        s = t.nextToken();
        if (s.charAt(0) == '\'') { // first '
            s = t.nextToken();
            if (s.charAt(0) == '\'') { // empty char
                throw new NumberFormatException();
            }
            else {
                argV.addElement(new Character(s.toCharArray()[0]));
            }
            break;
        }
    }
}
else { // argument is a number
    s = s.toLowerCase(); // ignore case

    if (s.indexOf("true") != -1) { //isBoolean
        argV.addElement(Boolean.valueOf("true"));
    }
    else if (s.indexOf("false") != -1) { //isBoolean
        argV.addElement(Boolean.valueOf("false"));
    }
    else if (s.indexOf(".") != -1) { // is Decimal
        if (s.indexOf("f") != -1) { // float
            argV.addElement(Float.valueOf(s));
        }
        else if (s.indexOf("d") != -1) { // double
            argV.addElement(Double.valueOf(s));
        }
        else argV.addElement(Double.valueOf(s));
    }
    else { // is int/short/byte/...
        int c = -1;
        if ((c = s.indexOf("b")) != -1) { // byte
            argV.addElement(Byte.valueOf(s.substring(0, s.length()-1)));
        }
        else if ((c = s.indexOf("s")) != -1) { // short
            argV.addElement(Short.valueOf(s.substring(0, s.length()-1)));
        }
        else if ((c = s.indexOf("l")) != -1) { // long
            argV.addElement(Long.valueOf(s.substring(0, s.length()-1)));
        }
        else if ((c = s.indexOf("i")) != -1) { // int
            argV.addElement(Integer.valueOf(s.substring(0, s.length()-1)));
        }
        else { // also int (if no appendix-letter exists)
            argV.addElement(Integer.valueOf(s));
        }
    }
}
}

catch (NumberFormatException nfe) {
    throw (new RuntimeException("Invalid or unknown datatype of argument nr "+count));
}
}

return argV;
}

} // eof CRule

```

1.3 CString

```

package com.strato.framework;

/*
 * @(#)CString.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

/**
 * A CString implements a mutable sequence of characters.
 * <p />
 * A CString is like a {@link String}, but can be modified like a
 * {@link StringBuffer} but not in its length. At any point in time it contains
 * some particular sequence of characters; an index sequence based on an
 * alphabet CAlphabet. The content of the sequence can be changed through certain
 * method calls.
 * <p />
 * The principal operations on a <code>CString</code> are the same as the
 * union of these of classes {@link String} and {@link StringBuffer}. So
 * because of this similarity, the use of this new String class should be as
 * easy as possbile.
 * <p />
 * In addition to this, there exists some access methods to some CString
 * typical features like an alphabet {@link CAlphabet} and an identifier (ID)
 * <p />
 * Finally CString implements the {@link IString} Interface, to get the
 * content as a {@link String} or a CharacterArray.
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.CAlphabet
 * @see java.lang.String
 * @see java.lang.StringBuffer
 * @since STRATO 1.0
 */

public class CString implements IString, java.io.Serializable {

    /** The array containing the current text as indexes of its corresponding alphabet */
    protected byte[] str;

    /** Reference to the correspoding CAlphabet instance */
    protected CAlphabet alphabet;

    /** Id-Reference to brand the input string */
    protected String id;

    /**
     * Initializes a newly created <code>CString</code> object so that it
     * represents an empty character sequence based on the standard alphabet
     * {@link com.strato.lib.alphabet.CAlphaNumeric}
     */
    public CString() {
        this("", CAlphabet.getInstanceByName(CAlphabet.DEFAULT_ALPHABET));
    }

    /**
     * Initializes a newly created <code>CString</code> object so that it
     * represents an empty character sequence based on an alphabet
     *
     * @param alphabet a <code>CAlphabet</code>.
     */
    public CString(CAlphabet alphabet) {
        this("", alphabet);
    }

    /**
     * Initializes a newly created <code>CString</code> object so that it

```

```

* represents the same sequence of characters as the argument based
* on the standard alphabet {@link com.strato.lib.alphabet.CAlphaNumeric}
*
* @param string a <code>String</code>.
*/
public CString(String string) {
    this(string, CAlphabet.getInstanceByName(CAlphabet.DEFAULT_ALPHABET));
}

/**
 * Constructs a CString filled with joker characters based on the
 * standard alphabet {@link com.strato.lib.alphabet.CAlphaNumeric}
 *
* @param length the initial capacity/length.
* @exception NegativeArraySizeException if the <code>length</code>
* argument is less than <code>0</code>.
*/
public CString(int len, CAlphabet alphabet) {
    this(null, len, alphabet);
}

/**
 * Initializes a newly created <code>CString</code> object so that it
 * represents the same sequence of characters as the argument based
 * on a given alphabet
 *
* @param string a <code>String</code>.
* @param alphabet a <code>CAlphabet</code>.
*/
public CString(String string, CAlphabet alphabet) {
    this(string, string.length(), alphabet);
}

/* The real constructor -> called by all others except of one */
private CString(String string, int length, CAlphabet alphabet) {
    if (length < 0)
        throw new NegativeArraySizeException("length must be >= 0");

    this.str = new byte[length]; // allocate Stringspace
    this.alphabet = alphabet;
    this.id = null;
    if (string == null)
        java.util.Arrays.fill(this.str, CAlphabet.JOKER_INDEX); // fill with JOKERS
    else {
        char[] ch = string.toCharArray(); // extract characters
        for(int i=0; i<this.str.length; i++)
            this.str[i] = this.alphabet.getIndex(ch[i]);
    }
}

/* Initializes a newly created <code>CString</code> object so that it
 * represents the byte sequence according to a given alphabet
 *
* @param seq a <code>byte[]</code> array of indexes.
* @param alphabet a <code>CAlphabet</code> instance.
*/
public CString(byte[] seq, CAlphabet alphabet) {
    if (seq == null)
        throw new NegativeArraySizeException("length must be >= 0");
    this.str = new byte[seq.length];
    System.arraycopy(seq, 0, this.str, 0, seq.length); // make a copy!!!
    this.alphabet = alphabet;
    this.id = null;

    // check if choosen index array seq is filled with valid characters
    // if not an ENoSuchCharacterException is thrown
    int size = alphabet.getSize();
    for(int i=0; i<this.str.length; i++) {
        if ((this.str[i] < 0) || (this.str[i] >= size))
            throw new ENoSuchCharacterException("Your sequence does contain invalid indexes");
    }
}

/**
 * Returns the assigned alphabet
 *
* @return the assigned alphabet

```

```

/*
public CALphabet getAlphabet() {
    return this.alphabet;
}

/**
 * Returns the assigned id
 *
 * @return the assigned id
 */
public String getId() {
    return this.id;
}

/**
 * Sets the desired id value
 *
 * @param value the value to be set
 */
public void setId(String id) {
    this.id = id;
}

/**
 * Returns the length of this CString.
 * The length is equal to the number of 16-bit Unicode characters
 * in the string.
 *
 * @return the length of the sequence of characters represented by this object.
 */
public int length() {
    return str.length();
}

/**
 * Returns the alphabet entry index of a character at the specified position in
 * the string. A position ranges from <code>0</code> to <code>length() - 1</code>.
 * The first character of the sequence is at position <code>0</code>,
 * the next at position <code>1</code>, and so on, as for array indexing.
 *
 * @param pos the position of the character in the string.
 * @return the alphabet entry index at the specified index of this string.
 *         The first character is at index <code>0</code>.
 * @exception StringIndexOutOfBoundsException if the <code>pos</code>
 *         argument is negative or not less than the length of this
 *         string.
 */
public byte getIndexAt(int pos) {
    if ((pos < 0) || (pos >= str.length()))
        throw new StringIndexOutOfBoundsException("pos must be in [0,"+(str.length-1)+"]");

    return str[pos];
}

/**
 * Sets the alphabet entry index of a character at the specified position in the
 * string. A position ranges from <code>0</code> to <code>length() - 1</code>.
 * The first character of the sequence is at position <code>0</code>,
 * the next at position <code>1</code>, and so on, as for array indexing.
 *
 * @param pos the position of the character.
 * @param indexvalue the alphabet entry index
 * @exception StringIndexOutOfBoundsException if the <code>pos</code>
 *         argument is negative or not less than the length of this
 *         string.
 */
public void setIndexAt(int pos, byte indexvalue) {
    if ((pos < 0) || (pos >= str.length()))
        throw new StringIndexOutOfBoundsException("pos must be in [0,"+(str.length-1)+"]");

    str[pos] = indexvalue;
}

/**
 * Returns the character at the specified position. A position ranges
 * from <code>0</code> to <code>length() - 1</code>. The first character
 * of the sequence is at position <code>0</code>, the next at position
 * <code>1</code>, and so on, as for array indexing.

```

```

/*
 * @param    pos   the position of the character.
 * @return   the character at the specified index of this string.
 *           The first character is at index <code>0</code>.
 * @exception StringIndexOutOfBoundsException  if the <code>pos</code>
 *           argument is negative or not less than the length of this
 *           string.
 */
public char getCharAt(int pos) {
    if ((pos < 0) || (pos >= str.length))
        throw new StringIndexOutOfBoundsException("pos must be in [0,"+(str.length-1)+"]");

    return alphabet.chars[ str[pos] ];
}

/**
 * Sets the character at the specified position. A position ranges
 * from <code>0</code> to <code>length() - 1</code>. The first character
 * of the sequence is at position <code>0</code>, the next at position
 * <code>1</code>, and so on, as for array indexing.
 *
 * @param    pos   the position of the character.
 * @return   the character at the specified index of this string.
 *           The first character is at index <code>0</code>.
 * @exception StringIndexOutOfBoundsException  if the <code>pos</code>
 *           argument is negative or not less than the length of this
 *           string.
 * @exception ENoSuchCharacterException  if the <code>char</code>
 *           argument is not a member of the alphabet assigned to this string
 */
public void setCharAt(int pos, char ch) {
    if ((pos < 0) || (pos >= str.length))
        throw new StringIndexOutOfBoundsException("pos must be in [0,"+(str.length-1)+"]");

    str[pos] = alphabet.getIndexof(ch);
}

/**
 * Returns the weight of a character at the specified position. The weight is
 * specified in the assigned alphabet CALphabet
 *
 * @param    pos   the position of the character.
 * @return   the weight of the character at the specified index of this
 *           string. The first character is at index <code>0</code>.
 * @exception StringIndexOutOfBoundsException  if the <code>pos</code>
 *           argument is negative or not less than the length of this
 *           string.
 */
public final double getWeightAt(int pos) {
    if ((pos < 0) || (pos >= str.length))
        throw new StringIndexOutOfBoundsException("pos must be in [0,"+(str.length-1)+"]");

    return alphabet.weights[ str[pos] ];
}

/**
 * Returns the prefix of a given lenght
 *
 * @param    len    The length of the prefix
 */
public CString prefix(int len) {
    return substring(0, len);
}

/**
 * Returns the suffix of a given lenght
 *
 * @param    len    The length of the suffix
 */
public CString suffix(int len) {
    return substring(str.length-len, str.length);
}

/**
 * Returns the suffix at a given position
 *
 * @param    startPos   The first character position of the suffix
 */

```

```

public CString suffixAt(int startPos) {
    return substring(startPos, str.length());
}

/* overwritten method from Object */
public String toString() {
    return getAsString();
}

private boolean comp(byte[] a, byte[] b) {
    if ((a == null) || (b == null)) throw new NullPointerException();
    if (a.length != b.length) return false;
    for(int i=0; i<a.length; i++) {
        if (a[i] != b[i]) return false;
    }
    return true; // all elements are equal
}

/* overwritten method from Object */
public boolean equals(Object anObject) {
    if (anObject instanceof CString) {
        CString aString= (CString)anObject;
        return ( comp(this.str, aString.str)
                  && this.getAlphabet().equals(aString.getAlphabet())
                );
    }
    return false;
}

/**
 * Returns the string as a index array (of bytes) corresponding to the
 * assigned alphabet. Warning: If you modify the array, you directly modify
 * the String itself! Therefore to modify it without affection to the original
 * string copy it first!!!
 *
 * @return      The string as index array of type byte
 */
public byte[] getAsByteArray() {
    return str;
}

/* Implementation of interface IString */
public String getAsString() {
    return new String(getAsCharArray());
}

/* Implementation of interface IString */
public char[] getAsCharArray() {
    int len = length();
    char[] charlist = new char[len];
    for(int i=0; i<len; i++) {
        charlist[i] = getCharAt(i);
    }
    return charlist;
}

/* from here we duplicate the java.lang.String interface */

/** see {@link java.lang.String}, same functionality but for CString ***/
public char charAt(int pos) {
    return this.getCharAt(pos);
}

/** see {@link java.lang.String}, same functionality but for CString ***/
public CString append(CString str) {
    int otherLen = str.length(); int thisLen = this.length();
    if (this.getAlphabet() != str.getAlphabet()) return null;
    if (otherLen == 0) return this;

    byte[] res = new byte[thisLen+otherLen];
    System.arraycopy(this.getAsByteArray(), 0, res, 0, thisLen);
    System.arraycopy(str.getAsByteArray(), thisLen, res, thisLen, otherLen);
    return new CString(res, this.getAlphabet());
}

/** see {@link java.lang.String}, same functionality but for CString ***/
public CString substring(int beginPos, int endPos) {

```

```
int len = endPos - beginPos;

if (beginPos < 0) {
    throw new StringIndexOutOfBoundsException(beginPos);
}
if (endPos > str.length) {
    throw new StringIndexOutOfBoundsException(endPos);
}
if (beginPos > endPos) {
    throw new StringIndexOutOfBoundsException(len);
}

byte[] b = new byte[len];
System.arraycopy(this.str, beginPos, b, 0, len);

return new CString(b, this.alphabet);
}

/** see {@link java.lang.String}, same functionality but for CString */
public char[] toCharArray() {
    return this.getAsCharArray();
}

/** see {@link java.lang.String}, same functionality but for CString */
public static CString concat(CString s1, CString s2) {
    return s1.append(s2);
}

} // eof CString
```

1.4 CStringStreamable

```

package com.strato.framework;

/*
 * @(#)CStringStreamable.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.lib.alphabet.CUserDefined;
import java.io.*;

/**
 * The CStringStreamable is a concrete implementation of IStreamable
 * <p />
 * The CStringStreamable class can be used to manage the streaming behavoir
 * of CStrings
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.IStreamable
 * @since STRATO 1.0
 */
public class CStringStreamable implements IStreamable{

    /* nof of characters printed per line */
    private final int CHARS_PER_LINE = 60;

    /* the String to be streamed */
    private CString string;

    /**
     * Creates an string streamable
     *
     * @param string    The stringName to be streamed
     */
    public CStringStreamable(CString string) {
        this.string = string;
    }

    /**
     * Returns the string we work with
     *
     * @return    The string to be streamed
     */
    public CString getString() { return this.string; }

    /**
     * Sets the string we would work with
     *
     * @param string    The string to be streamed
     */
    public void setString(CString string) { this.string = string; }

    /* implements the IStreamable interface */
    public void loadFromStream(InputStream ins) throws Exception {
        InputStreamReader isr = null;
        BufferedReader br = null;
        try {
            isr = new InputStreamReader(ins);
            br = new BufferedReader(isr);

            StringBuffer buf = new StringBuffer();
            String line = br.readLine();
            while (line != null) {
                buf.append(line);
                if ( (line.equals("")) || (line.charAt(0) == '!') ) break; // options section begins
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (isr != null) {
                try {
                    isr.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (br != null) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

```

```

        int idxR = line.indexOf('\r'); if (idxR != -1) buf.deleteCharAt(idxR);
        int idxN = line.indexOf('\n'); if (idxN != -1) buf.deleteCharAt(idxN);
        int len = line.length();

        line = br.readLine();
    }

    String str = buf.toString();
    CALphabet alpha = null;
    double[] weights = null;
    String id = null;

    if (line == null) { // no more additional information available
        alpha = CALphabet.getInstanceByName(CALphabet.DEFAULT_ALPHABET);
        this.string = new CString(str, alpha);
    }
    else {
        line = br.readLine();

        while (line != null) {
            if (line.equalsIgnoreCase(";alphabet;")) {
                line = br.readLine();
                if (line.charAt(0) == CALphabet.JOKER) { // is User Def
                    alpha = CALphabet.getInstanceByName(CALphabet.USERDEFINED_ALPHABET);
                    ((CUserDefined)alpha).setContentByString(line);
                }
                else { // is Predefined
                    alpha = CALphabet.getInstanceByName(line);
                }
            }
            else if (line.equalsIgnoreCase(";weights;")) {
                line = br.readLine();
                weights = CStringUtils.StringToDoubleArray(line);
            }
            else if (line.equalsIgnoreCase(";id;")) {
                line = br.readLine();
                id = line;
            }

            line = br.readLine();
        } // eof switch-while
    }

    if (alpha == null) {
        alpha = CALphabet.getInstanceByName(CALphabet.DEFAULT_ALPHABET);
    }
    this.string = new CString(str, alpha);

    if ((weights != null) && (weights.length == alpha.getSize())) {
        string.getAlphabet().setWeights(weights);
    }
    if (id != null) {
        string.setId(id);
    }
}
finally {
    br.close();
    isr.close();
}
}

/* implements the IStreamable interface */
public void saveToStream(OutputStream outs) throws Exception {
    PrintWriter pr = null;
    try {
        pr = new PrintWriter(outs);
        String str = this.string.getAsString();

        while (str.length() > CHARS_PER_LINE) {
            pr.println(str.substring(0, CHARS_PER_LINE));
            pr.flush();
            str = str.substring(CHARS_PER_LINE, str.length());
        }
        pr.println(str); // insert final characters
        pr.println(); // insert clear line
        pr.flush();

        //now we have to Store the Alphabet
    }
}

```

```
pr.println(";alphabet");
CAlphabet alpha = string.getAlphabet();
if (alpha instanceof CUserDefined) {
    pr.println(string.getAlphabet().getAsString());
}
else { // print classname
    pr.println(alpha.getClass().getName());
}

// now the weights
pr.println(";weights");
pr.println(CStringUtil.arrayToString(string.getAlphabet().getWeights()));

// now ev. the ID
if (string.getId() != null) {
    pr.println(";id");
    pr.println(string.getId());
}

pr.flush();
}
finally {
    //    pr.close(); // don't do this cause of malfunction in some cases
}
}

} // eof CStringStreamable
```

1.5 ENoSuchCharacterException

```
package com.strato.framework;

/*
 * @(#)ENoSuchCharacterException.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

/**
 * Thrown when you try to use a character not defined in the actual alphabet
 * <p>
 * Typical causes are the setXXX methods of CAlphabet and CString.
 *
 * @author Daniel Emmenegger
 * @see com.strato.framework.CAlphabet
 * @see com.strato.framework.CString
 * @version 1.0 2002/03/03
 * @since STRATO 1.0
 */
public class ENoSuchCharacterException extends RuntimeException{

    /**
     * Constructs an <code>ENoSuchCharacterException</code> without message
     */
    public ENoSuchCharacterException() {
        super();
    }

    /**
     * Constructs an <code>ENoSuchCharacterException</code> with a
     * detailed message.
     *
     * @param msg Descriptive message
     */
    public ENoSuchCharacterException(String msg) {
        super(msg);
    }
} // eof ENoSuchCharacterException
```

1.6 IAccess

```

package com.strato.framework;

/*
 * @(#) IAccess.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

/**
 * An interface to get access to the two interfaces of type {@link IInput} and
 * {@link IOutput}. They provide itself access to the current input data and
 * the current output devices.
 * <p />
 * It acts as bridge between the STRATO system and written Tests (Plug-ins).
 * <p />
 * This is the only way for Tests to get access to the STRATO system - because
 * all tests are loaded dynamically as {@link java.lang.Class} object by their
 * classname during runtime.
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.IInput
 * @see com.strato.framework.IOutput
 * @see com.strato.framework.test.ITestInfo
 * @see com.strato.framework.test.CSimpleTestInfo
 * @since STRATO 1.0
 */

public interface IAccess {

    /**
     * Returns the interface to alle necessary inputs for tests
     * (e.g. the current Input String)
     *
     * @return A handle to access any sort of available input data
     */
    public IInput getInputs();

    /**
     * Returns the Interface to alle necessary ouput devices
     * (e.g. the Console)
     *
     * @return A handle to access any sort of available output devices
     */
    public IOutput getOutputs();

} // eof IAccess

```

1.7 IInput

```
package com.strato.framework;

/*
 * @(#)IInput.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

/**
 * An interface to get access to the current input data.
 * <p />
 * At the moment there is only one input type: the actual Input String as an
 * instance of class <code>CString</code>.
 * <p />
 * This "three-interface-structure" {@link IAccess}, {@link IInput} and
 * {@link IOutput} might seem to be some sort of overkill at present, but it
 * is intended to be extended in future releases (multiple inputs, ...) and
 * should therefore be down-compatible to previous versions.
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.IAccess
 * @see com.strato.framework.IOutput
 * @see com.strato.framework.test.CSimpleTestInfo
 * @since STRATO 1.0
 */

public interface IInput {

    /**
     * Returns the instance of the currently used input string
     *
     * @return An input string
     */
    public abstract CString getInputString();

} // eof IInput
```

1.8 IOutput

```

package com.strato.framework;

/*
 * @(#)IOutput.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

/**
 * An interface to get access to the current output devices.
 * <p />
 * At the moment there is only one ouput device: the console.
 * <p />
 * This "three-interface-structure" {@link IAccess}, {@link IInput} and
 * {@link IOutput} might seem to be some sort of overkill at present, but it
 * is intended to be extended in future releases (graphical devices, ...) and
 * should therefore be down-compatible to previous versions
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.IAccess
 * @see com.strato.framework.IOutput
 * @see com.strato.framework.test.CSimpleTestInfo
 * @since STRATO 1.0
 */

public interface IOutput {

    /**
     * Provides the PrintWriter Object, which is directed to the console
     */
    public java.io.PrintWriter Console();

    /**
     * Provides functionality to write to the statusbar
     */
    public void setStatus(String text);

    /**
     * Provides functionality to set the percentage value of to the progressbar
     */
    public void setProgress(int value);

} // eof IOutput

```

1.9 IPersistence

```

package com.strato.framework;

/*
 * @(#)IPersistence.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

/**
 * This interface is used to make any data (input, output, intermediate results,
 * ...) persistent. For example in a local file, via TCP/IP stream on a remote database.
 * <p />
 * This persistency includes also sheets of papers resulting as output of
 * printining devices like laser printers or inkjets.
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.gui.input.CAbstractInput
 * @see com.strato.gui.output.CAbstractOutput
 * @since STRATO 1.0
 */

public interface IPersistence {

    /**
     * Clears the current content of the media displaying the data.
     * E.g. in case of a text component tc: -> tc.setText("");
     */
    public void clear();

    /**
     * Opens a resource and get the data.
     * Default implementation should be a {@link java.awt.FileDialog} whereby the
     * user has to overwrite the template method loadFromStream(java.io.InputStream)
     * See {@link com.strato.gui.output.CAbstractOutput} as an example.
     */
    public void open();

    /**
     * Saves date persistently.
     * Default implementation should be a {@link java.awt.FileDialog} whereby the
     * user has to overwrite the template method saveToStream(java.io.OutputStream).
     * See {@link com.strato.gui.output.CAbstractOutput} as an example.
     */
    public void save();

    /**
     * Prints data on a printing device.
     * Default implementation should be a {@link java.awt.FileDialog} whereby the
     * user has to overwrite the template method drawContent(java.awt.print.PrinterJob)
     * The implementing class of this interface should further implement the
     * {@link java.awt.print.Printable} interface.
     */
    public void print();

} // eof IPersistence

```

1.10 IStreamable

```
package com.strato.framework;

/*
 * @(#)IStreamable.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import java.io.*;

/**
 * This interface serves as common access method to load and store data
 * <p />
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.gui.output.CAbstractOutput
 * @since STRATO 1.0
 */

public interface IStreamable {

    /**
     * Loads the data from a stream
     *
     * @param ins The InputStream where the data comes from
     */
    public void loadFromStream(InputStream ins) throws Exception;

    /**
     * Saves the data to a stream
     *
     * @param outs The OutputStream where the data goes to
     */
    public void saveToStream(OutputStream outs) throws Exception;

} // eof IStreamable
```

1.11 IString

```
package com.strato.framework;

/*
 * @(#)IString.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

/**
 * A class that implements the <em>IString</em> interface if it can specify
 * its current content by a {@link java.lang.String} or an array of <bold>char</bold>.
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see      com.strato.framework.CString
 * @see      com.strato.framework.CAlphabet
 * @since   STRATO 1.0
 */
public interface IString {

    /**
     * Return the current content of the underlying object as a java.lang.String
     *
     * @return The current content as String
     */
    public String getAsString();

    /**
     * Returns the current content of the underlying object as char array
     *
     * @return The current content as char[]
     */
    public char[] getAsCharArray();

} // eof IString
```

2 com.strato.framework.run

2.1 CAbstractCase

```

package com.strato.framework.run;

/*
 * @(#)CAbstractCase.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

/**
 * An abstract class providing some interface to launch a test.
 * <p>
 * Inherit all the methods of this class and implement at least the
 * <code>launch()</code> method.
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.run.CSTSICase
 * @see com.strato.framework.run.CRunner
 * @since STRATO 1.0
 */
public abstract class CAbstractCase {

    /** A name to specify the case */
    protected String name;

    /**
     * Constructor
     *
     * @param name      any description/name of the case
     */
    public CAbstractCase(String name) {
        this.name = name;
    }

    /**
     * This method is called immediatly before the test ist launched
     * It might be overwritten by subclasses
     */
    protected void setUp() { /* empty */ }

    /**
     * This method is called immediatly after the test finished (if and only
     * if no Exception occurred!)
     * It might be overwritten by subclasses
     */
    protected void tearDown() { /* empty */ }

    /**
     * This method should call the selected test
     * @exception throws an exception of type InvocationTargetException if the
     *             started test raised any sort of Exception
     */
    public abstract void launch() throws Exception;
}

```

2.2 CRunner

```

package com.strato.framework.run;

/*
 * @(#)CRunner.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import java.io.*;
import java.lang.reflect.*;

/**
 * A singleton class to run tests.
 * <p>
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.run.CAbstractCase
 * @see com.strato.framework.run.CSTSICase
 * @since STRATO 1.0
 */
public final class CRunner {

    /** test is initialized */
    public static final int IDLE = 0;
    /** test is running */
    public static final int RUNNING = 1;
    /** test is paused */
    public static final int PAUSED = 2;
    /** test is stopped / terminated */
    public static final int STOPPED = 3;

    /** maximal amount of test we can run at the same time (in parallel) */
    protected static final int MAX_THREADS = 5;

    /** singleton instance of the runner */
    private static CRunner runner = null;

    /** Target writer to publish Runtime Exceptions raised by a test */
    private PrintWriter msgWriter;

    /** the currently assigned case of test */
    protected CAbstractCase suiteClass;

    /** the array to hold the thread references */
    protected Thread[] thread;

    /** the array to keep track on every test's state */
    protected int[] state;

    /**
     * Creates a CRunner
     */
    private CRunner() {
        this.msgWriter = new PrintWriter(System.err);
        this.suiteClass = new CSTSICase("dummy", null, 0);
        this.thread = new Thread[MAX_THREADS];
        this.state = new int[MAX_THREADS];
        java.util.Arrays.fill(this.state, IDLE);
    }

    /**
     * The only possibility to get an instance of CRunner
     */
    public static CRunner getRunner() {
        if (runner == null) {
            runner = new CRunner();
        }
    }
}

```

```

        return runner;
    }

    /**
     * Sets the target writer, where we want the messages displayed
     *
     * @param msgWriter The target writer to print on
     */
    public void setMessageWriter(PrintWriter msgWriter) {
        this.msgWriter = msgWriter;
    }

    /**
     * Returns the state of test (thread) with a given identifier
     *
     * @param tID The test (thread) ID we would like to query
     * @return Returns the state of the desired test
     */
    public synchronized int getSTATE(int tID) {
        return state[tID];
    }

    /**
     * Sets the case for this Runner
     *
     * @param suiteClass The case of test to be assigned to the runner
     */
    public void setCase(CAbstractCase suiteClass) {
        this.suiteClass = suiteClass;
    }

    /**
     * Initializes a test (thread)
     *
     * @param tID The test (thread) ID we would like to initialize
     */
    public void INIT(int tID) {
        this.state[tID] = IDLE;
        this.thread[tID] = new Thread(new MyRunnable(tID), "TestID ["+tID+"]");
    }

    /**
     * Starts an initialized test (thread) asynchronously
     *
     * @param tID The test (thread) ID we would like to start
     */
    public void START(int tID) {
        if (this.state[tID] == IDLE) {
            this.state[tID] = RUNNING;
            this.thread[tID].start();
        }
        else {
            msgWriter.println("Previous test was stopped and must be reinitialized "+
                "with INIT before we can reuse it -> check implementation");
        }
    }

    /**
     * Continues a previously suspended test (thread)
     *
     * @param tID The test (thread) ID we would like to restart
     */
    public void CONTINUE(int tID) {
        this.state[tID] = RUNNING;
        this.thread[tID].resume();
    }

    /**
     * Stops a running test (thread)
     *
     * @param tID The test (thread) ID we would like to stop
     */
    public void STOP(int tID) {
        this.state[tID] = STOPPED;
        this.thread[tID].stop();
    }

    /**

```

```
* Pauses (suspends) a running test (thread)
*
* @param tID    The test (thread) ID we would like to pause
*/
public void PAUSE(int tID) {
    this.state[tID] = PAUSED;
    this.thread[tID].suspend();
}

/**
 * Waits until the termination of a running test (thread)
*
* @param tID    The test (thread) ID we would like to wait for
*/
public void JOIN(int tID) {
    try {
        this.thread[tID].join();
        if (this.state[tID] == RUNNING) this.state[tID] = IDLE;
    }
    catch (InterruptedException ex) { }
}

/* launch the test */
private void go() {
    try {
        suiteClass.setUp();
        suiteClass.launch();
        suiteClass.tearDown();
    }
    catch (InvocationTargetException ite) {
        if (ite.getTargetException() instanceof ThreadDeath) {
            throw new ThreadDeath();
        }
        else {
            msgWriter.println("<< Test raised "+ite.getTargetException()+" -> check implementation
>>");
            msgWriter.flush();
        }
    }
    catch (Exception e) {
        System.err.println("ERROR while setUp or tearDown of a test: "+e.getMessage());
    }
}

class MyRunnable implements Runnable {

    private int ID;

    public MyRunnable(int ID) {
        this.ID = ID;
    }

    public void run() {
        CRunner.getRunner().go();
    }
}

} // eof CRunner
```

2.3 CSTSICase

```

package com.strato.framework.run;

/*
 * @(#)CSTSICase.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.test.*;
import java.lang.reflect.*;

/**
 * A class for a Single Test with Single Input (STSI).
 * <p>
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.run.CAbstractCase
 * @see com.strato.framework.run.CRunner
 * @since STRATO 1.0
 */
public class CSTSICase extends CAbstractCase {

    /** a reference to the descriptors */
    protected TDescriptorShortcuts sc = null;

    /** the currently selected algorithm variant */
    protected int algoID = 0;

    /**
     * Constructor to create a case for the <code>CRunner</code> class. At the
     * moment (Version 1.0) this is the only available case class.
     *
     * @param name    any name for the case
     * @param testHandle    the handle to access the test
     * @param algoID    the currently selected algorithm variant
     */
    public CSTSICase(String name, ITestInfo testHandle, int algoID) {
        super(name);
        this.sc = new TDescriptorShortcuts(testHandle);
        this.algoID = algoID;
    }

    /* inherited */
    protected void setUp() {
        // for example:
        // ASSERT: sc.ti != null;
    }

    /* inherited */
    public void launch() throws Exception {
        try {
            sc.aDesc[algoID].getRunMethod().invoke(sc.refDesc.getInstance(), null);
        }
        catch (IllegalAccessException iae) {
            System.err.println(iae.getMessage());
        }
    }

    /* inherited */
    protected void tearDown() {
        // for example:
        // ASSERT: sc.refDesc.getReferenceTo(sc.refDesc.RESULT) [algoID] != null;
    }
} // CSTSICase

```

3 com.strato.framework.test

3.1 CAlgorithmDescriptor

```

package com.strato.framework.test;

/*
 * @(#)CAlgorithmDescriptor.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import java.lang.reflect.*;

/**
 * A CAlgorithmDescriptor describes a particular algorithm that a STRATO Test
 * provides to the system.
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see    com.strato.framework.test.CGenericDescriptor
 * @see    com.strato.framework.test.ITestInfo
 * @since  STRATO 1.0
 */
public class CAlgorithmDescriptor extends CGenericDescriptor {

    private Method runMethod = null;
    private CResultDescriptor resDesc = null;

    /**
     * Constructs a <code>CAlgorithmDescriptor</code>
     * You have to specify to which class its belongs to: for example "MyText.class"
     *
     * @param ownerClass    The class the algorithm belongs to.
     * @param name          The programmatic name of this algorithm.
     * @param displayName   The name of this algorithm to be displayed.
     */
    public CAlgorithmDescriptor(Class ownerClass, String name, String displayName)
        throws EConventionException {
        setName(name);
        setDisplayName(displayName);
        try {
            runMethod = CConvention.findMethod(ownerClass, name, 0);
        }
        catch (Exception ex) {
            throw new EConventionException("the run method '"+name+"' can't be found");
        }
    }

    /**
     * Returns the method that this CAlgorithmDescriptor encapsualtes.
     *
     * @return    The low-level description of the algorithm method
     */
    public Method getRunMethod() {
        return this.runMethod;
    }

    /**
     * Returns the a CResultDescriptor instance, that exactly describes the type of
     * the produced result. This allows to get acquire information than
     * <code>resultInstance.getClass</code> can provide.
     *
     * @return    Descriptor giving additional information about the result type
     */
    public CResultDescriptor getResultDescriptor() {
        return this.resDesc;
    }
}

```

```
}

/**
 * Sets the a CResultDescriptor instance, that exactly describes the type of
 * the produced result.
 *
 * @param      Descriptor specifying additional information about the result type
 */
public void setResultDescriptor(CResultDescriptor resultDesc) {
    this.resDesc = resultDesc;
}

} // eof CAlgorithmDescriptor
```

3.2 CConvention

```

package com.strato.framework.test;

/*
 * @(#)CConvention.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import java.lang.reflect.*;
import java.security.*;

/**
 * The CConvention class provides a library of tools to learn about
 * the algorithms, parameters, ... supported by a target STRATO Test.
 * <p>
 * For each of those kinds of information, the CConvention class will
 * separately help to analyze the tests's class and superclasses looking for
 * only explicit information (provided by any test through the different
 * CxxxDescriptors) and uses that information to get access to the target test.
 * <p>
 * Each test class has to provide this explicit information about itself by an
 * implementation of the ITestInfo interface.
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.test.CTestDescriptor
 * @see com.strato.framework.test.CAlgorithmDescriptor
 * @see com.strato.framework.test.CParameterDescriptor
 * @since STRATO 1.0
 */

public final class CConvention {

    /* Cache of Class.getDeclaredMethods: */
    private static java.util.Hashtable declaredMethodCache = new java.util.Hashtable();

    private static synchronized Method[] getPublicDeclaredMethods(Class clz) {
        // Looking up Class.getDeclaredMethods is relatively expensive,
        // so we cache the results.
        final Class fclz = clz;
        Method[] result = (Method[])declaredMethodCache.get(fclz);
        if (result != null) {
            return result;
        }

        // We have to raise privilege for getDeclaredMethods
        result = (Method[]) AccessController.doPrivileged(new PrivilegedAction() {
            public Object run() {
                return fclz.getDeclaredMethods();
            }
        });
    }

    // Null out any non-public methods.
    for (int i = 0; i < result.length; i++) {
        Method method = result[i];
        int mods = method.getModifiers();
        if (!Modifier.isPublic(mods)) {
            result[i] = null;
        }
    }
    // Add it to the cache.
    declaredMethodCache.put(clz, result);
    return result;
}

/**

```

```

* Internal support for finding a target methodName on a given class.
*/
private static Method internalFindMethod(Class start, String methodName, int argCount) {

    // For overridden methods we need to find the most derived version.
    // So we start with the given class and walk up the superclass chain.
    for (Class cl = start; cl != null; cl = cl.getSuperclass()) {
        Method methods[] = getPublicDeclaredMethods(cl);
        for (int i = 0; i < methods.length; i++) {
            Method method = methods[i];
            if (method == null) {
                continue;
            }
            // skip static methods.
            int mods = method.getModifiers();
            if (Modifier.isStatic(mods)) {
                continue;
            }
            if (method.getName().equals(methodName) &&
                method.getParameterTypes().length == argCount) {
                return method;
            }
        }
    }

    // Now check any inherited interfaces. This is necessary both when
    // the argument class is itself an interface, and when the argument
    // class is an abstract class.
    Class ifcs[] = start.getInterfaces();
    for (int i = 0 ; i < ifcs.length; i++) {
        Method m = internalFindMethod(ifcs[i], methodName, argCount);
        if (m != null) {
            return m;
        }
    }
}

return null;
}

/**
 * Internal support for finding a target methodName with a given
 * parameter list on a given class.
 */
private static Method internalFindMethod(Class start, String methodName,
                                         int argCount, Class args[]) {

    // For overridden methods we need to find the most derived version.
    // So we start with the given class and walk up the superclass chain.
    for (Class cl = start; cl != null; cl = cl.getSuperclass()) {
        Method methods[] = getPublicDeclaredMethods(cl);
        for (int i = 0; i < methods.length; i++) {
            Method method = methods[i];
            if (method == null) {
                continue;
            }
            // skip static methods.
            int mods = method.getModifiers();
            if (Modifier.isStatic(mods)) {
                continue;
            }
            // make sure method signature matches.
            Class params[] = method.getParameterTypes();
            if (method.getName().equals(methodName) &&
                params.length == argCount) {
                boolean different = false;
                if (argCount > 0) {
                    for (int j = 0; j < argCount; j++) {
                        if (params[j] != args[j]) {
                            different = true;
                            continue;
                        }
                    }
                }
                if (different) {
                    continue;
                }
            }
        }
    }
}
}

```

```

}

// Now check any inherited interfaces.  This is necessary both when
// the argument class is itself an interface, and when the argument
// class is an abstract class.
Class ifcs[] = start.getInterfaces();
for (int i = 0 ; i < ifcs.length; i++) {
    Method m = internalFindMethod(ifcs[i], methodName, argCount);
    if (m != null) {
        return m;
    }
}

return null;
}

/**
 * Find a target methodName on a given class.
 *
 * @param cls The corresponding class
 * @param methodName The programmatic name of the method to be found
 * @param argCount The number of arguments
 *
 * @return The low-level description of the method data
 */
public static Method findMethod(Class cls, String methodName, int argCount)
    throws EConventionException {
    if (methodName == null) {
        return null;
    }

    Method m = internalFindMethod(cls, methodName, argCount);
    if (m != null ) {
        return m;
    }

    // We failed to find a suitable method
    throw new EConventionException("No method \\" + methodName +
                                    "\\" with " + argCount + " arg(s)");
}

/**
 * Find a target methodName with specific parameter list on a given class.
 *
 * @param cls The corresponding class
 * @param methodName The programmatic name of the method to be found
 * @param argCount The number of arguments
 * @param args[] A specific parameter list (the arguments)
 * @return The low-level description of the method data
 */
public static Method findMethod(Class cls, String methodName, int argCount,
    Class args[]) throws EConventionException {
    if (methodName == null) {
        return null;
    }

    Method m = internalFindMethod(cls, methodName, argCount, args);
    if (m != null ) {
        return m;
    }

    // We failed to find a suitable method
    throw new EConventionException("No method \\" + methodName +
                                    "\\" with " + argCount + " arg(s) of matching types.");
}

} // eof CConvention

```

3.3 CCtrlCenter

```

package com.strato.framework.test;

/*
 * @(#)CCtrlCenter.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.*;
import java.lang.reflect.*;

/**
 * The CCtrlCenter class provides some reflection functionality to access
 * the parameters within a test.
 * <p>
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @since STRATO 1.0
 */

public class CCtrlCenter {

    private TDescriptorShortcuts sc = null;

    /**
     * Constructor to create an instance of the control center
     *
     * @param    shortcuts    The shortcuts to access a test
     */
    public CCtrlCenter(TDescriptorShortcuts shortcuts) {
        this.sc = shortcuts;
    }

    /* Parameter IO */

    /**
     * Reads the description of a parameter with index <code>curParam</code>
     */
    public String readParameterQuery(int curParam) {
        return sc.pDesc[curParam].getDisplayName() + "?";
    }

    /**
     * Reads the value of the desired parameter
     *
     * @param    curParam    id of the desired parameter
     *           (see {@link CParameterDescriptor})
     */
    public String readParameterValue(int curParam) {
        String res = null;
        try {
            res = sc.pDesc[curParam].getReadMethod().invoke(
                sc.refDesc.getInstance(), null).toString();
        }
        catch (InvocationTargetException ite) {
            System.err.println("Could not find appropriate method get access to the parameter");
        }
        catch (IllegalAccessException iae) {
            System.err.println("System cannot access parameter");
        }
        return res;
    }

    /**
     * Writes the value to the desired parameter
     */

```

```
* @param curParam id of the desired parameter (see {@link CParameterDescriptor})
* @param value the value to be set in string representation
*/
public void writeParameterValue(int curParam, String value) {
    try {
        Class clz = sc.pDesc[curParam].getParameterType();

        sc.pDesc[curParam].getWriteMethod().invoke(sc.refDesc.getInstance(),
            new Object[] {CStringUtils.convertStringToObject(value, clz)})
    }
}
catch (InvocationTargetException ite) {
    System.err.println("Could not find appropriate method get access to the parameter");
}
catch (IllegalAccessException iae) {
    System.err.println("System cannot access parameter");
}
catch (Exception e) {
    System.err.println("Error while converting parameter-string to object");
}
}

} // eof CCtrlCenter
```

3.4 CExactMatchingTestInfo

```

package com.strato.framework.test;

/*
 * @(#)CExactMatchingTestInfo.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.*;

/**
 * This is an abstract support class to make it easier for people to write tests
 * doing string searches. We define a parameter called "pattern" and its
 * corresponding setter and getter methods.
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see      com.strato.framework.test.ITestInfo
 * @see      com.strato.framework.test.CSimpleTestInfo
 * @see      com.strato.tests.BoyerMoore
 * @since    STRATO 1.0
 */
public abstract class CExactMatchingTestInfo extends CSimpleTestInfo {

    /** parameter: the string we look for */
    protected String pattern;

    /**
     * Constructor for Exact Matching Tests (abstract class)
     */
    public CExactMatchingTestInfo(IAccess access) {
        super(access);
    }

    /** Algorithm: Does pattern exist? */
    public abstract void searchExists();

    /** Algorithm: How often does the pattern appear? */
    public abstract void searchCount();

    /** Algorithm: First occurrence of the pattern? */
    public abstract void searchFirst();

    /** Algorithm: All occurrences of the pattern? */
    public abstract void searchAll();

    public CParameterDescriptor[] getParameterDescriptors() {
        try {
            CParameterDescriptor pattern =
                new CParameterDescriptor(this.getClass(), "pattern", String.class);
            pattern.setDisplayName("pattern P");
            pattern.setShortDescription("the string (pattern) we look for");
            // is of type: java.lang.String

            CParameterDescriptor[] ret = {pattern};
            return ret;
        }
        catch (EConventionException ce) {
            throw new Error(ce.toString());
        }
    }

    public CALgorithmDescriptor[] getAlgorithmDescriptors() {
        try {
            CALgorithmDescriptor algoExists =
                new CALgorithmDescriptor(this.getClass(), "searchExists", "Does the pattern P
exist?");
        }
    }
}

```

```
    CALgorithmDescriptor algoCount =
        new CALgorithmDescriptor(this.getClass(), "searchCount", "How often does the pattern P
appear?");

    CALgorithmDescriptor algoFirst =
        new CALgorithmDescriptor(this.getClass(), "searchFirst", "First occurrence of the
pattern P?");

    CALgorithmDescriptor algoAll =
        new CALgorithmDescriptor(this.getClass(), "searchAll", "All occurrences of the pattern
P?");

    CALgorithmDescriptor[] ret = {algoExists, algoCount, algoFirst, algoAll};
    return ret;
}
catch (EConventionException ce) {
    throw new Error(ce.toString());
}
}

public void reset() {
    // do here anything to reset your algorithm
    super.reset();
    this.pattern = "dummy";
}

public String getPattern() {
    return pattern;
}

public void setPattern(String pattern) {
    this.pattern = pattern;
}

} // eof CExactMatchingTestInfo
```

3.5 CGenericDescriptor

```

package com.strato.framework.test;

/*
 * @(#)CGenericDescriptor.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

/**
 * The CGenericDescriptor class is the common baseclass for {@link CTestDescriptor},
 * {@link CAlgorithmDescriptor}, and {@link CParameterDescriptor}.
 * <p>
 * It supports some common information that can be set and retrieved from
 * any of the introspection descriptors.
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.test.ITestInfo
 * @since STRATO 1.0
 */
public abstract class CGenericDescriptor {

    private static final String DESC_NOT_AVAIL = "description not available";

    private String name;
    private String displayName;
    private String shortDescription;

    /**
     * Constructs a <code>CGenericDescriptor</code>.
     */
    public CGenericDescriptor() {
        // do nothing
    }

    /**
     * Gets the programmatic name of this feature.
     *
     * @return The programmatic name of the test/algorithm/parameter
     */
    public String getName() {
        return name;
    }

    /**
     * Sets the programmatic name of this feature.
     *
     * @param name The programmatic name of the test/algorithm/parameter
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Gets the localized display name of this feature.
     *
     * @return The localized display name for the test/algorithm/parameter.
     *         This defaults to the same as its programmatic name from getName.
     */
    public String getDisplayName() {
        if (displayName == null) {
            return getName();
        }
        return displayName;
    }

    /**
     * Sets the localized display name of this feature.
     */

```

```
*  
* @param displayName  The localized display name for the  
*   test/algorithm/parameter.  
*/  
public void setDisplayName(String displayName) {  
    this.displayName = displayName;  
}  
  
/**  
 * Gets the short description of this feature.  
*  
* @return A localized short description associated with this  
*   test/algorithm/parameter. This defaults to be the display name.  
*/  
public String getShortDescription() {  
    if (shortDescription == null) {  
        return DESC_NOT_AVAIL;  
    }  
    return shortDescription;  
}  
  
/**  
 * You can associate a short descriptive string with a feature. Normally  
* these descriptive strings should be less than about 40 characters.  
* @param text A (localized) short description to be associated with  
* this test/algorithm/parameter.  
*/  
public void setShortDescription(String text) {  
    shortDescription = text;  
}  
  
/* override Objectimpl. */  
public String toString() {  
    return getDisplayName();  
}  
} // eof CGenericDescriptor
```

3.6 CParameterDescriptor

```

package com.strato.framework.test;

/*
 * @(#)CParameterDescriptor.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import java.lang.reflect.*;
import java.beans.*;

/**
 * A CParameterDescriptor describes one parameter that a STRATO Test
 * exports via a pair of accessor methods.
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see      com.strato.framework.test.CGenericDescriptor
 * @see      com.strato.framework.test.ITestInfo
 * @since    STRATO 1.0
 */
public class CParameterDescriptor extends CGenericDescriptor {

    private Class parameterType;
    private Method readMethod;
    private Method writeMethod;

    /**
     * Constructs a CParameterDescriptor for a parameter that follows
     * the standard Java convention by having getFoo and setFoo
     * accessor methods. Thus if the argument name is "fred", it will
     * assume that the writer method is "setFred" and the reader method
     * is "getFred" (or "isFred" for a boolean parameter). Note that the
     * parameter name should start with a lower case character, which will
     * be capitalized in the method names.
     *
     * @param parameterName The programmatic name of the parameter.
     * @param testClass The Class object for the target test. For
     *                 example com.strato.test.MyTest.class.
     * @exception EConventionException if an exception occurs during
     *                                 introspection.
     */
    public CParameterDescriptor(Class ownerClass, String parameterName, Class parameterType)
        throws EConventionException {
        if (parameterName == null || parameterName.length() == 0) {
            throw new EConventionException("bad parameter name");
        }
        setName(parameterName);
        String base = capitalize(parameterName);

        // Since there can be multiple setter methods but only one getter
        // method, find the getter method first so that you know what the
        // property type is. For booleans, there can be "is" and "get"
        // methods. If an "is" method exists, this is the official
        // reader method so look for this one first.
        try {
            readMethod = CConvention.findMethod(ownerClass, "is" + base, 0);
        }
        catch (Exception getterExc) {
            // no "is" method, so look for a "get" method.
            readMethod = CConvention.findMethod(ownerClass, "get" + base, 0);
        }
        Class params[] = { readMethod.getReturnType() };
        writeMethod = CConvention.findMethod(ownerClass, "set" + base, 1, params);

        findParameterType();
    }
}

```

```

    /**
     * Gets the Class object for the parameter.
     *
     * @return The Java type info for the property. Note that
     * the "Class" object may describe a built-in Java type such as "int".
     * The result may be "null" if this is an indexed property that
     * does not support non-indexed access.
     * <p>
     * This is the type that will be returned by the ReadMethod.
     */
    public Class getParameterType() {
        return parameterType;
    }

    /**
     * Gets the method that should be used to read the parameter value.
     *
     * @return The method that should be used to read the property value.
     * May return null if the property can't be read.
     */
    public Method getReadMethod() {
        return readMethod;
    }

    /**
     * Sets the method that should be used to read the parameter value.
     *
     * @param getter The new getter method.
     */
    public void setReadMethod(Method setter) throws EConventionException {
        readMethod = setter;
        findParameterType();
    }

    /**
     * Gets the method that should be used to write the property value.
     *
     * @return The method that should be used to write the property value.
     * May return null if the property can't be written.
     */
    public Method getWriteMethod() {
        return writeMethod;
    }

    /**
     * Sets the method that should be used to write the property value.
     *
     * @param setter The new setter method.
     */
    public void setWriteMethod(Method setter) throws EConventionException {
        writeMethod = setter;
        findParameterType();
    }

    private void findParameterType() throws EConventionException {
        try {
            parameterType = null;
            if (readMethod != null) {
                if (readMethod.getParameterTypes().length != 0) {
                    throw new EConventionException("bad read method arg count");
                }
                parameterType = readMethod.getReturnType();
                if (parameterType == Void.TYPE) {
                    throw new EConventionException("read method " +
                        readMethod.getName() + " returns void");
                }
            }
            if (writeMethod != null) {
                Class params[] = writeMethod.getParameterTypes();
                if (params.length != 1) {
                    throw new EConventionException("bad write method arg count");
                }
                if (parameterType != null && parameterType != params[0]) {
                    throw new EConventionException("type mismatch between read and write methods");
                }
                parameterType = params[0];
            }
        }
    }
}

```

```
        }
        catch (EConventionException ex) {
            throw ex;
        }
    }

    static String capitalize(String s) {
        if (s.length() == 0) {
            return s;
        }
        char chars[] = s.toCharArray();
        chars[0] = Character.toUpperCase(chars[0]);
        return new String(chars);
    }

} // eof CParameterDescriptor
```

3.7 CReferenceDescriptor

```

package com.strato.framework.test;

/*
 * @(#)CReferenceDescriptor.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

/**
 * This Descriptor provides access to the test instance itself an some
 * special variables (e.g. results)
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.test.CGenericDescriptor
 * @see com.strato.framework.test.ITestInfo
 * @since STRATO 1.0
 */
public class CReferenceDescriptor {

    /** type of the instance reference, you can also use getInstance() instead */
    public final static int INSTANCE = 0;
    /** type of the result reference */
    public final static int RESULT = 1;

    private Object instance;
    private Object[] result;

    /**
     * Constructs a <code>CReferenceDescriptor</code>
     * You have to specify to which class its belongs to. e.g. "MyText.class"
     *
     * @param ownerClass  The class the test represents.
     * @param instance    The ref to the instance the descriptor belongs to
     * @param result      A ref to the calculated results (represented as an Object[])
     */
    public CReferenceDescriptor(Class ownerClass, Object instance, Object[] result) {
        this.instance = instance;
        this.result = result;
    }

    /**
     * Returns a reference a desired instance.
     *
     * @param type      the kind of reference we would like to be returned
     * @return         The concrete reference to the instance
     */
    public Object getReferenceTo(int type) {
        switch (type) {
            case INSTANCE: return this.instance;
            case RESULT: return this.result;
            default: return null;
        }
    }

    /**
     * Returns the instance the descriptor belongs to. Shortcut of getReferenceTo()
     *
     * @return         The concrete instance of a test implementing this descriptor
     */
    public Object getInstance() {
        return this.instance;
    }
} // eof CReferenceDescriptor

```

3.8 CSimpleTestInfo

```

package com.strato.framework.test;

/*
 * @(#)CSimpleTestInfo.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.*;

/**
 * This is a support class to make it easier for people to provide
 * TBeanInfo information an to handle the access of data (input and output).
 * <p />
 * It defaults to providing "noop" information, and can be selectively
 * overriden to provide more explicit information on chosen topics.
 * When the introspector sees the "noop" values, it will trap if the searched
 * information is mandatory for a test and not supplementary like information
 * for exemplal on parameters, if test has none of them.
 * <p />
 * Further more it defines some shortcuts to the IAccess interface which
 * should still held updated by calling the <code>update()</code> method
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see      com.strato.framework.test.ITestInfo
 * @see      com.strato.framework.test.CExactMatchingTestInfo
 * @since   STRATO 1.0
 */
public abstract class CSimpleTestInfo implements ITestInfo {

    /** the handle to the access interface */
    protected IAccess access;

    /** the handle to all input sources */
    protected IInput inputs;
    /** the handle to all output devices */
    protected IOutput outputs;

    /** used input shortcut for the 'input string' */
    protected CString inString;
    /** used input shortcut for the 'input alphabet' */
    protected CAphabet inAlphabet;

    /** a place to store the results */
    protected Object[] result;

    /**
     * This class is abstract and can therefore not be instanstialted
     * The constructor does only some shortcut stuff.
     * This method calls <code>reset()</code> from ITestInfo
     *
     * @param access      handle to access all the necessary data
     */
    public CSimpleTestInfo(IAccess access) {
        setAccess(access);

        // to raise Exception during Test launch if result is not instantiated!!!
        this.result = null;

        // reset all the stuff!!!
        reset();
    }

    /**
     * This class is abstract and can therefore not be instanstialted
     * The constructor does only some shortcut stuff.
     */
}

```

```

 * This method calls <code>reset()</code> from ITestInfo
 */
public CSimpleTestInfo() {
    this(null);
}

/**
 * Returns the access handle
 *
 * @return      access handle
 */
public IAccess getAccess() {
    return this.access;
}

/**
 * Sets the access handle if you called the standard constructor or
 * you would like to change the current access to the system
 *
 * @param access      access handle to be set
 */
public void setAccess(IAccess access) {
    this.access = access;
    if (access != null) update();
}

/**
 * Updates all shortcuts, should be called as first function in a top-level
 * method if you use the shortcuts.
 * The two shortcuts <code>inputs</code> and <code>outputs</code> can be outdated,
 * if you do not you use them like <code>access.getInputs()</code>.
 */
protected void update() {
    inputs = access.getInputs();
    outputs = access.getOutputs();

    inString = inputs.getInputString();
    if (inString != null) {
        inAlphabet = inString.getAlphabet();
    }
    else { // announce to user with an exception
        throw new EInvalidUserInputException("Your input string is invalid!");
    }
}

/**
 * Simply returns a minimal, but not <code>null</code> implementation
 */
public CTestDescriptor getTestDescriptor() {
    CTestDescriptor testDesc =
        new CTestDescriptor(this.getClass(), "Test: [" + this.getClass().getName() + "]");
    try { testDesc.setResetable(testDesc.RESET_DEFAULTNAME); }
    catch (EConventionException ce) {
        try { testDesc.setResetable(null); }
        catch (EConventionException cexc) { }
    }
    return testDesc;
}

/**
 * Simply returns the a standard, but not <code>null</code> implementation
 */
public CReferenceDescriptor getReferenceDescriptor() {
    return new CReferenceDescriptor(this.getClass(), this, this.result);
}

/**
 * Simply returns <code>null</code>
 */
public CAAlgorithmDescriptor[] getAlgorithmDescriptors() {
    return null;
}

/**
 * Simply returns <code>null</code> -> means: no parameters
 */
public CParameterDescriptor[] getParameterDescriptors() {

```

```
    return null;
}

/**
 * Has to be overwritten by any subclass giving resetting functionality to
 * your STRATO test! Especially you have to instantiate the <code>result</code>
 * field, with the correct numbers of array elements (= nof Algorithms)
 */
public void reset() {
    // do nothing at the moment
}

} // CSimpleTestInfo
```

3.9 CTestDescriptor

```

package com.strato.framework.test;

/*
 * @(#)CTestDescriptor.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import java.lang.reflect.*;

/**
 * A CTestDescriptor provides global information about a "test",
 * including its Java owner class, its displayName, etc.
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see    com.strato.framework.test.CGenericDescriptor
 * @see    com.strato.framework.test.ITestInfo
 * @since STRATO 1.0
 */
public class CTestDescriptor extends CGenericDescriptor {

    /** The default name of the reset method provided by STRATO Tests */
    public static String RESET_DEFAULTNAME = "reset";

    private Class ownerClass;
    private Method resetMethod;

    /**
     * Constructs a <code>CTestDescriptor</code>
     * You have to specify to which class its belongs to. e.g. "MyTest.class"
     *
     * @param ownerClass    The class the test represents.
     * @param instance      The instance the descriptor belongs to
     * @param displayName   The name of this algorithm to be displayed.
     */
    public CTestDescriptor(Class ownerClass, String displayName) {
        this.ownerClass = ownerClass;
        setName(ownerClass.getName());
        setDisplayName(displayName);
        try { setResetable(RESET_DEFAULTNAME); } catch (EConventionException ce) { }
    }

    /**
     * Returns if the test is resetable
     *
     * @return    availability to be resetable
     */
    public boolean isResetable() {
        return (this.resetMethod != null);
    }

    /**
     * Sets the availability to be resetable on/off
     *
     * @param name    the programmatic name of the reset method, if name == null ->
     *                test is not resetable!
     */
    public void setResetable(String name) throws EConventionException {
        String methodname = RESET_DEFAULTNAME; // default

        if (name != null) {
            if (!name.equals("")) methodname = name;
            this.resetMethod = CConvention.findMethod(ownerClass, methodname, 0);
        }
        else {
            this.resetMethod = null;
        }
    }
}

```

```
}

/***
 * Return the reset() Method of the Test
 *
 * @return the reset method handle or null if not available
 */
public Method getResetMethod() {
    return this.resetMethod;
}

} // eof CTestDescriptor
```

3.10 EConventionException

```
package com.strato.framework.test;

/*
 * @(#)EConventionException.java    1.0    2002/03/03
 *
 * Title:      STRING-AnaLysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

/**
 * Thrown when an exception happens during Introspection of a class that
 * should provide methods in a special way (naming conventions).
 * <p>
 * Typical causes include not being able to map a string class name
 * to a Class object, not being able to resolve a string method name,
 * or specifying a method name that has the wrong type signature for
 * its intended use.
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.test.CConvention
 * @since STRATO 1.0
 */
public class EConventionException extends Exception {

    /**
     * Constructs an <code>EConventionException</code> without message
     */
    public EConventionException() {
        super();
    }

    /**
     * Constructs an <code>EConventionException</code> with a
     * detailed message.
     *
     * @param msg Descriptive message
     */
    public EConventionException(String msg) {
        super(msg);
    }
} // eof EConventionException
```

3.11 EInvalidUserInputException

```
package com.strato.framework.test;

/*
 * @(#)EInvalidUserInputException.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

/**
 * Thrown by a test, when the user input (delivered to the test) is not in a given
 * range of valid values.
 * <p>
 * Typical causes are a negative lenght of the input string or similar things.
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @since STRATO 1.0
 */
public class EInvalidUserInputException extends RuntimeException {

    /**
     * Constructs an <code>EInvalidUserInputException</code> without message
     */
    public EInvalidUserInputException() {
    }

    /**
     * Constructs an <code>EInvalidUserInputException</code> with a
     * detailed message.
     *
     * @param msg Descriptive message
     */
    public EInvalidUserInputException(String msg) {
        super(msg);
    }
} // eof EInvalidUserInputException
```

3.12 EUpdateException

```
package com.strato.framework.test;

/*
 * @(#)EUpdateException.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

/**
 * Thrown when an exception happens during Displaying any result data of a test,
 * which might not be available or be <code>null</code>.
 * <p>
 * Typical causes are not already started tests or made changes in the GUI which
 * are used to preprocess the output (e.g. selecting another alphabet).
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @since STRATO 1.0
 */
public class EUpdateException extends Exception {

    /**
     * Constructs an <code>EUpdateException</code> without message
     */
    public EUpdateException() {
        super();
    }

    /**
     * Constructs an <code>EConventionException</code> with a
     * detailed message.
     *
     * @param msg Descriptive message
     */
    public EUpdateException(String msg) {
        super(msg);
    }
} // eof EUpdateException
```

3.13 IGraphicResult

```

package com.strato.framework.test;

/*
 * @(#)IGraphicResult.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

/**
 * This interface has to be implemented by each test that would like to produce
 * some graphical output.
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.test.IResult
 * @since STRATO 1.0
 */
public interface IGraphicResult extends IResult {

    /**
     * This callback method can be used to setup the data to be displayed next
     *
     * @param container This is the container to drawing has to fit into. Use
     *                  the <code>getInsets()</code> method to get its extent.
     * @param algoID   The ID of the algorithm from which the result
     *                  should be displayed
     * @exception EUpdateException if the necessary result data is not already
     *                             available or is <code>null</code>
     */
    public void setupPainting(java.awt.Container container, int algoID) throws EUpdateException;

    /**
     * This callback method does the raw drawing job based on the updated
     * values (managed by the test class)
     *
     * @param g        The graphics handle to paint on
     * @param algoID  The ID of the algorithm from which the result
     *                should be displayed
     * @exception EUpdateException if the necessary result data is not already
     *                            available or is <code>null</code>
     */
    public void paint(java.awt.Graphics g, int algoID) throws EUpdateException;
}

// eof IGraphicResult

```

3.14 IResult

```
package com.strato.framework.test;

/*
 * @(#)IResult.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

/**
 * Empty generic interface, does noting at the moment.
 * There are two interfaces "subclassed" of this generic one
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see      com.strato.framework.test.ITextResult
 * @see      com.strato.framework.test.IGraphicResult
 * @since   STRATO 1.0
 */

public interface IResult {

} // eof IResult
```

3.15 ITestInfo

```

package com.strato.framework.test;

/*
 * @(#)ITestInfo.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

/**
 * A STRATO Test implementor is obligated to provide explicit information
 * about his test.
 * <p />
 * A test implementor needs to provide a complete set of explicit information.
 * Only the {@link CParameterDescriptor} should return <code>null</code> if it has none
 * of them.
 * <p />
 * See also the SimpleTestInfo class which provides a convenient
 * "noop" base class for STRATO test classes, which you have override
 * for those specific places.
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.test.CTestDescriptor
 * @see com.strato.framework.test.CAlgorithmDescriptor
 * @see com.strato.framework.test.CParameterDescriptor
 * @see com.strato.framework.test.CSimpleTestInfo
 * @since STRATO 1.0
 */
public interface ITestInfo {

    /**
     * Returns a CTestDescriptor instance
     *
     * @return Descriptor giving information about the test itself
     */
    public CTestDescriptor getTestDescriptor();

    /**
     * Returns a CReferenceDescriptor instance. Use it with care and readonly!
     *
     * @return Descriptor giving access to (even protected/private) values
     */
    public CReferenceDescriptor getReferenceDescriptor();

    /**
     * Returns a CAlgorihtm Descriptor instance
     *
     * @return Descriptor Array giving information about provided algorithms
     */
    public CAlgorithmDescriptor[] getAlgorithmDescriptors();

    /**
     * Returns a CAlgorihtm Descriptor instance
     *
     * @return Descriptor Array giving information about supported parameters
     *         (concerning the algorithms)
     */
    public CParameterDescriptor[] getParameterDescriptors();

} // eof TestInfo

```

3.16 ITextResult

```
package com.strato.framework.test;

/*
 * @(#)ITextResult.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

/**
 * This interface has to be implemented by each test that would like to produce
 * some textual output.
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.test.IResult
 * @since STRATO 1.0
 */
public interface ITextResult extends IResult{

    /**
     * This callback method does the raw writing job based on the updated
     * values (managed by the test class). Use HTML Tags <html> and </html> to
     * format the text: e.g. tc.setText("<html>Result = <i>result[i]</i></html>");
     *
     * @param tc      The text component to write into
     * @param algoID  The ID of the algorithm from which the result
     *                should be displayed
     * @exception EUpdateException if the necessary result data is not already
     *                            available or is <code>null</code>
     */
    public void write(javax.swing.text.JTextComponent tc, int algoID) throws EUpdateException;

} // eof ITextResult
```

4 com.strato.lib

4.1 CRandom

```

package com.strato.lib;

/*
 * @(#)CRandom.java 1.0 2002/03/03
 *
 * Title: STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright: Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company: ETH Zurich, Switzerland / Department of Computer Science
 * Homepages: http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import java.util.Random;

/**
 * This class provides different generators for (pseudo)random numbers
 * <p />
 * 1.) Java Standard (class java.util.Random) -> STD
 * 2.) Linear congruential: aNew = (aOld * b + 1) mod m -> LC
 * 3.) Additive congruential: -> AC
 *      i) shift all a[j] cyclic, increase pointer: j++
 *      ii) aNew = a[j] = (a[31] + a[54]) mod m
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @since STRATO 1.0
 */
public final class CRandom {

    /** uses java.util.Random */
    public static final int JAVA_STANDARD = 0;
    /** a generator based on the linear congruential method */
    public static final int LINEAR_CONGRUENTIAL = 1;
    /** a generator based on the additive congruential method */
    public static final int ADDITIVE_CONGRUENTIAL = 2;

    /* size needed for the AC method */
    private static final int ARRAY_SIZE = 55;

    /* a number which satifies m < (MAX_INT / 2) */
    private static final int m = 100000000;
    /* m1 = sqrt(m) -> used to avoid integer overflow during mult(...) */
    private static final int m1 = 10000;
    /* any positive number ending up in ..21 */
    private static final int b = 314155821;

    /* the instance for the STD variant */
    private static Random random = null;

    /* the cycling pointer in the AC variant */
    private int curJ = 0;
    /* an array to store the last ARRAY_SIZE numbers for the AC variant */
    private int[] a = new int[ARRAY_SIZE];

    /* the upperbound of the produced random number */
    private int upTo;

    /* the currently chosen type (kind) of generator e.g. STD, AC, LC */
    private int kind = 0;

    /**
     * Constructor to initialize the random generator
     */
    public CRandom() {
        this(m);
    }
}

```

```

/**
 * Constructor to initialize the random generator
 *
 * @param upTo The upper bound for the range the generator has to produce a
 *              random number: [0..upTo[
 */
public CRandom(int upTo) {
    this.upTo = upTo;
    random = new Random();
    setKind(JAVA_STANDARD, Math.abs((int)System.currentTimeMillis()));
}

/**
 * Call this to change the type of random number generator
 *
 * @param kind the type of the generator (use the constants in CRandom)
 * @param seed the initial seed for the generator
 */
public void setKind(int kind, int seed) {
    this.kind = kind;
    setSeed(seed);
}

/**
 * Call this to change the type of random number generator
 *
 * @param seed the initial seed for the currently selected generator
 */
public void setSeed(int seed) {
    seed = Math.abs(seed);      // avoid Exceptions ;-
    switch(this.kind) {
        case LINEAR_CONGRUENTIAL: initLC(seed); break;
        case ADDITIVE_CONGRUENTIAL: initAC(seed); break;
        case JAVA_STANDARD:
        default: initSTD(seed); break;
    }
}

/**
 * Return the currently selected type of random number generator
 *
 * @return the currently selected type
 */
public int getKind() {
    return this.kind;
}

/**
 * Sets the upperbound of the result range: [0..upTo[
 *
 * @param upTo the upper bound to be set
 */
public void setUpperBound(int upTo) {
    this.upTo = upTo;
}

/**
 * Returns the upperbound of the result range: [0..upTo[
 *
 * @return the currently set upper bound
 */
public int getUpperBound() {
    return this.upTo;
}

/**
 * Returns the next random number of type int in the range [0..upTo[
 *
 * @return the next integer (pseudo)random number
 */
public int random() {
    int res = 0;
    switch(kind) {
        case LINEAR_CONGRUENTIAL: res = randLC(); break;
        case ADDITIVE_CONGRUENTIAL: res = randAC(); break;
        case JAVA_STANDARD:
        default: res = randSTD(); break;
    }
}

```

```

        return res;
    }

/* initialize the Standard Java variant */
private void initSTD(int seed) {
    random.setSeed(seed);
}

/* initialize the Linear Congruential variant */
private void initLC(int seed) {
    a[0] = seed;
}

/* initialize the Additive Congruential variant */
private void initAC(int seed) {
    a[0] = seed;
    for(int j=1; j<ARRAY_SIZE; j++)
        a[j] = (mult(b,a[j-1])+1) % m;
    curJ = ARRAY_SIZE-1;
}

/* helper method to avoid integer overflow while mult(a,b)!!! */
private int mult(int p, int q) {
    int p1 = p / m1; int p0 = p % m1;
    int q1 = q / m1; int q0 = q % m1;
    return (((p0*q1+p1*q0) % m1)*m1+p0*q0) % m;
}

/* called by random() in case of STD */
private int randSTD() {
    return random.nextInt(this.upTo);
}

/* called by random() in case of LC */
private int randLC() {
    a[0] = (mult(a[0],b)+1) % m;
    return ((a[0] / m1)*upTo) / m1;
}

/* called by random() in case of AC */
private int randAC() {
    curJ = (curJ+1) % 55;
    a[curJ] = (a[(curJ+32) % 55] + a[(curJ+54) % 55]) % m;
    return ((a[curJ] / m1)*upTo) / m1;
}

/** only for testing reasons and to demonstrate the use of the class */
public static void main(String args[]) {
    CRandom r = new CRandom(2000);
    r.setKind(r.LINEAR_CONGRUENTIAL, 99);
    for(int i=0; i<100; i++) {
        System.out.print(r.random()+" ");
    }
    System.out.println();
    r.setKind(r.ADDITIVE_CONGRUENTIAL, 99);
    for(int i=0; i<100; i++) {
        System.out.print(r.random()+" ");
    }
}

} // eof CRandom

```

5 com.strato.lib.alphabet

5.1 CLowerCase

```

package com.strato.lib.alphabet;

/*
 * @(#)CLowerCase.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.CAlphabet;

/**
 * Alphabet of all latin lowercase letters
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.CAlphabet
 * @see com.strato.framework.CString
 * @since STRATO 1.0
 */

public class CLowerCase extends CAlphabet {

    static final char[] CHARSET =
    { JOKER,
        'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
        'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
        'u', 'v', 'w', 'x', 'y', 'z' };

    private static final String V = "vowel";
    private static final String C = "consonant";

    private static final String[] DESCRIPTIONS =
    { "JOKER",
        V, C, C, C, V, C, C, C, V, C,
        C, C, C, C, V, C, C, C, C, C,
        V, C, C, C, C, C };

    public CLowerCase() {
        super("Lower case letters: a..z", CHARSET);

        this.defaultDescriptions = this.DESCRIPTIONS;
        this.setDescriptions(this.DESCRIPTIONS);
    }

} // eof CLowerCase

```

5.2 CNotQuiteASCII

```

package com.strato.lib.alphabet;

/*
 * @(#)CNotQuiteASCII.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.CAlphabet;

/**
 * An alphabet that consists of quite alle US-ASCII Chars
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see      com.strato.framework.CAlphabet
 * @see      com.strato.framework.CString
 * @since    STRATO 1.0
 */

public final class CNotQuiteASCII extends CAlphabet {

    private static final int CHARSET_SIZE = 96; // incl JOKER

    static char[] CHARSET;
    private static String[] DESCRIPTIONS;

    static {
        buildCharset();
        buildDescriptions();
    }

    public CNotQuiteASCII() {
        super("ASCII", CHARSET);
        this.setDescriptions(this.DESCRIPTIONS);
    }

    private static void buildCharset() {
        byte[] set = new byte[CHARSET_SIZE];
        set[0] = (byte)CAlphabet.JOKER;
        for(int i=1; i<CHARSET_SIZE; i++) {
            set[i] = (byte)(31 + i);
        }
        try {
            CHARSET = (new String(set, "ISO-8859-1")).toCharArray();
        }
        catch (java.io.UnsupportedEncodingException use) {
            System.err.println("Not supported Encoding");
            CHARSET = new char[]{JOKER};
        }
    }

    private static void buildDescriptions() {
        DESCRIPTIONS = new String[CHARSET_SIZE];
        DESCRIPTIONS[0] = "JOKER";
        for(int i=1; i<CHARSET_SIZE; i++) {
            DESCRIPTIONS[i] = "ASCII: "+ (i+31);
        }
    }
} // eof CNotQuiteASCII

```

5.3 CProteine

```

package com.strato.lib.alphabet;

/*
 * @(#)CProteine.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.CAlphabet;

/**
 * Alphabet constisting of the 20 commonly occuring Amino Acid Residues
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see      com.strato.framework.CAlphabet
 * @see      com.strato.framework.CString
 * @since    STRATO 1.0
 */

public class CProteine extends CAlphabet {

    static final char[] CHARSET =
    { JOKER,
        'A', 'R', 'N', 'D', 'C', 'Q', 'E', 'G', 'H', 'I',
        'L', 'K', 'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V' };

    private static double[] WEIGHTS_AVERAGE =
    { 0.0,
        71.0788, 156.1876, 114.1039, 115.0886, 103.1448, 128.1308, 129.1155,
        57.0520, 137.1412, 113.1595, 113.1595, 128.1742, 131.1986, 147.1766,
        97.1167, 87.0782, 101.1051, 186.2133, 163.1760, 99.1326 };

    private static final String[] DESCRIPTIONS =
    { "JOKER",
        "Alanine", "Arginine", "Asparagine", "Aspartic Acid", "Cysteine",
        "Glutamine", "Glutamic Acid", "Glycine", "Histidine", "Isoleucine",
        "Leucine", "Lysine", "Methionine", "Phenylalanine", "Proline",
        "Serine", "Threonine", "Tryptophan", "Tyrosine", "Valine" };

    public CProteine() {
        super("Proteins", CHARSET);

        this.defaultWeights = this.WEIGHTS_AVERAGE;
        this.setWeights(this.WEIGHTS_AVERAGE);
        this.defaultDescriptions = this.DESCRIPTIONS;
        this.setDescriptions(this.DESCRIPTIONS);
    }

} // eof CProteine

```

5.4 CUserDefined

```

package com.strato.lib.alphabet;

/*
 * @(#)CUserDefined.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.CAlphabet;

import java.util.*;

/**
 * An alphabet the consists of the characters occurring in a given string
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.CAlphabet
 * @see com.strato.framework.CString
 * @since STRATO 1.0
 */

public class CUserDefined extends CAlphabet {

    static final char[] CHARSET =
        { JOKER, 'A', 'B', 'C' };

    public CUserDefined() {                      // Default implementation ABC
        super("User Def", CHARSET);
    }

    public CUserDefined(String displayName, String charsetString) {
        super(displayName, charsetString);
    }

    public void setContentByString(String charsetString) {
        this.chars = charsetString.toCharArray();
        this.initProperties();
    }

    /**
     * Extracts a charset out of a given string
     *
     * @param str      The string we extract all occurring characters as set
     */
    public static char[] extractCharset(String str) {
        List charList = Collections.synchronizedList(new ArrayList());
        charList.add(new Character(JOKER));
        for(int i=0; i<str.length(); i++) {
            Character c = new Character(str.charAt(i));
            if (!charList.contains(c)) {
                charList.add(c);
            }
        }
        Object[] list = charList.toArray();
        char[] res = new char[list.length];
        for(int i=0; i<res.length; i++){
            res[i] = ((Character)list[i]).charValue();
        }
        Arrays.sort(res,1,res.length-1);
        return res;
    }

    /**
     * Returns the char array (charset) as {@link java.lang.String}. See
     * <code>extractCharset</code> for further information.
     *
     * @param str      The string we extract all occurring characters as set
     */
}

```

```
/*
public static String extractCharsetAsString(String str) {
    return new String(extractCharset(str));
}

} // eof CUserDefined
```

6 com.strato.lib.ds

6.1 CSuffixTree

```

package com.strato.lib.ds;

/*
 * @(#)CSuffixTree.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.*;
import java.util.*;

/**
 * A generalized suffix tree implementation
 *
 * The algorithm is explained in the paper
 *   A Practical Suffix-Tree Implementation for String Searches
 *   by Bogdan Dorohonceanu and Craig Nevill-Manning
 *   published in Dr. Dobb's Journal, July 2000
 * <p>
 * superclasses are SuffixArray, SuffixTree
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @since STRATO 1.0
 */
public class CSuffixTree extends SuffixTree {

    /**
     * Creates a Suffixtree for one given CString
     *
     * @param string    the string we build the suffix tree from
     */
    public CSuffixTree(CString string) {
        super(100, (string.length() / 10 + 10), new CString[] {string}, false);
    }

    /**
     * Creates a Suffixtree for a given CString array
     *
     * @param strings   the string array we build the suffix tree from
     */
    public CSuffixTree(CString[] strings) {
        super(100, (strings[0].length() / 10 + 10), strings, false);
    }

    /* Section: Counting all substrings */
    //***** //

    long sum_;

    /**
     * Count all substrings in O(n)
     */
    public long countSubstrings() {
        sum_ = 0L;
        for (int child = child_[root_]; child > 0; child = sibling_[child]) {
            countSubstrings(child);
        }
        return sum_;
    }

    // Auxiliary method for counting all substrings into the tree

```

```

// by traversing the tree.
private void countSubstrings(int node) {
    // inner node
    if (child_[node] > 0) {
        long diff = (long) (index_[child_[node]] - index_[node]);
        sum_ += diff;
        for (int child = child_[node]; child > 0; child = sibling_[child]) {
            countSubstrings(child);
        }
    }
    // leaf -> do nothing
}

/* Section: Enumerating all substrings */
/*****************/
int[] p_ = null;

/**
 * Returns a {interval range} array for each suffix:
 */
public int[] enumerateSubstrings() {
    p_ = new int[length_];      // nof leaves
    java.util.Arrays.fill(p_, -1);

    for (int child = child_[root_]; child > 0; child = sibling_[child]) {
        enumerateSubstrings(0, 0, child);
    }

    return p_;
}

private void enumerateSubstrings(int totallength, int labellength, int node) {
    if (child_[node] > 0) { // inner node
        int first = index_[node];      int last = index_[child_[node]];
        labellength = last - first;
        totallength += labellength;

        for (int child = child_[node]; child > 0; child = sibling_[child]) {
            enumerateSubstrings(totallength, labellength, child);
            totallength = 0;      // already proceeded
        }
    }
    // leaf
    else {
        int suffix = -child_[node];
        int seqNo = findSequenceNo(suffix);          // is always == 0 !!!
        int seqSuffix = suffix - offset_[seqNo];

        // set tail lenght for a given suffix
        if (totallength == 0)
            p_[seqSuffix] = labellength;
        else
            p_[seqSuffix] = totallength;
    }
}

int beginIdx;

/**
 * Returns a pair {interval range, next node} array for each suffix, plus the
 * beginning index.
 */
public int[] enumerateSortedSubstrings() {
    p_ = new int[length_*2 + 1];      // nof leaves * 2 (pair entry) + beginning idx
    java.util.Arrays.fill(p_, -1);    // fill with "not filled" = -1

    beginIdx = -1;
    int prevIdx = -1;
    for (int child = child_[root_]; child > 0; child = sibling_[child]) {
        prevIdx = enumerateSortedSubstrings(0, 0, prevIdx, child);
    }

    p_[p_.length-1] = beginIdx;      // append the begin idx

    return p_;
}

```

```

    private int enumerateSortedSubstrings(int totallength, int labellength, int prevIdx, int
node) {
    if (child_[node] > 0) { // inner node
        int first = index_[node]; int last = index_[child_[node]];
        labellength = last - first;
        totallength += labellength;

        for (int child = child_[node]; child > 0; child = sibling_[child]) {
            prevIdx = enumerateSortedSubstrings(totallength, labellength, prevIdx, child);
            totallength = 0; // already proceeded
        }
        return prevIdx;
    }
    // leaf
    else {
        int suffix = -child_[node];
        int seqNo = findSequenceNo(suffix); // is always == 0 !!!
        int seqSuffix = suffix - offset_[seqNo];

        // set tail lenght for a given suffix
        if (totallength == 0)
            p_[seqSuffix*2] = labellength;
        else
            p_[seqSuffix*2] = totallength;

        // ev. set beginIdx
        if (beginIdx == -1) beginIdx = seqSuffix;

        // set pointer (in any case)
        if (prevIdx >= 0)
            p_[prevIdx*2 + 1] = seqSuffix;

        return seqSuffix; // return last node name (suffix start index)
    }
}

/** Section of old stuff -> very good for testing purposes ***/
//*****



/**
 * Returns a String array of all substrings
 */
public String[] enumerateSubstringsAsStringArray() {
    v_ = new Vector();
    for (int child = child_[root_]; child > 0; child = sibling_[child]) {
        enumerateSubstringsAsStringArray(child, "");
    }

    String[] res = new String[v_.size()];
    for (int i = 0; i < res.length; i++) {
        res[i] = (String)v_.elementAt(i);
    }
    v_ = null;
    return res;
}

private void enumerateSubstringsAsStringArray(int node, String prefix) {
    if (child_[node] > 0) {
        // inner node
        int first = index_[node]; int last = index_[child_[node]];
        int diff = last - first;
        String p = prefix + this.substring(index_[node], diff, false);
        for (int i=diff-1; i>=0; i--) {
            v_.add(new String(p.substring(0, p.length()-i)));
        }
        for (int child = child_[node]; child > 0; child = sibling_[child]) {
            enumerateSubstringsAsStringArray(child, p);
        }
    }
    // leaf
}
} // eof CSuffixTree

```

7 com.strato.lib.output

7.1 CBarPlotAssistant

```

package com.strato.lib.output;

/*
 * @(#)CBarPlotAssistant.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import java.awt.*;

/**
 * A class providing some very simple bar-plot functionality
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.lib.output.CPlotAssistant
 * @see com.strato.lib.output.CLinePlotAssistant
 * @since STRATO 1.0
 */
public class CBarPlotAssistant extends CPlotAssistant {

    /**
     * Creates a Barplot Assistant
     */
    public CBarPlotAssistant(Container container) {
        super(container);
    }

    public void paintContent(Graphics g) {
        super.paintContent(g);

        // draw line graph
        for (int i = 0; i < x.length; i++) {
            Color col = ((i%2) == 0) ? Color.blue : Color.red;
            g.setColor(col);
            int x0 = (int)(xpmint+i*xfactor);  int y0 = calcy(y[i]);
            int dx = (int)xfactor;  int dy = ypmax - y0;
            g.fillRect(x0, y0, dx, dy);
        }
    }
} // eof CBarPlotAssistant

```

7.2 CLinePlotAssistant

```
package com.strato.lib.output;

/*
 * @(#)CLinePlotAssistant.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import java.awt.*;

/**
 * A class providing some very simple line-plot functionality
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see      com.strato.lib.output.CPlotAssistant
 * @see      com.strato.lib.output.CBarPlotAssistant
 * @since    STRATO 1.0
 */
public class CLinePlotAssistant extends CPlotAssistant {

    /**
     * Creates a Lineplot Assistant
     */
    public CLinePlotAssistant(Container container) {
        super(container);
    }

    public void paintContent(Graphics g) {
        super.paintContent(g);

        // get first point
        int xp = calcx(x[0]);
        int yp = calcy(y[0]);

        // draw line graph
        g.setColor(color);
        for (int i = 1; i < x.length; i++) {
            int xpNext = calcx(x[i]);
            int ypNext = calcy(y[i]);
            g.drawLine(xp, yp, xpNext, ypNext);
            xp = xpNext;
            yp = ypNext;
        }
    }
} // eof CLinePlotAssistant
```

7.3 CPlotAssistant

```

package com.strato.lib.output;

/*
 * @(#)CPlotAssistant.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import java.awt.*;

/**
 * A superclass of all plot assistants
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see      com.strato.lib.output.CBarPlotAssistant
 * @see      com.strato.lib.output.CLinePlotAssistant
 * @since   STRATO 1.0
 */
public abstract class CPlotAssistant {

    transient protected double xfactor, yfactor;
    transient protected int xpmin, ypmin, xpmax, ypmax;

    /** lower bound of the x-axis */
    protected double minX;
    /** lower bound of the y-axis */
    protected double minY;
    /** upper bound of the x-axis */
    protected double maxX;
    /** upper bound of the y-axis */
    protected double maxY;

    /** global array of the x values */
    protected double[] x;
    /** global array of the y values */
    protected double[] y;
    /** global color to paint with */
    protected Color color;

    /** a reference to the container where we place the chart */
    protected Container container;

    /**
     * Creates an plot assistant instance with respect to the container size
     *
     * @param container The container where we place the chart
     */
    public CPlotAssistant(Container container) {
        this.container = container;
        setOutputBounds(0.0,0.0,1.0,1.0);
    }

    /**
     * Sets the viewing window of the koordinate system
     *
     * @param minX    Lower bound of the x-axis
     * @param maxX    Upper bound of the x-axis
     * @param minY    Lower bound of the y-axis
     * @param maxY    Upper bound of the y-axis
     */
    public void setOutputBounds(double minx, double miny, double maxx, double maxy) {
        minX = minx;  minY = miny;  maxX = maxx;  maxY=maxy;
    }

    /**
     * Calculate the local koordinate with respect to the viewing window
     */
}

```

```

        * @param    xp      The x value in world coordinates
        */
protected int calcx(double xp) {
    return (int)((xp-minX) * xfactor + xpmin);
}

/**
 * Calculate the local koordinate with respect to the viewing window
 *
 * @param    yp      The y value in world coordinates
 */
protected int calcy(double yp) {
    int ypnt = (int)((yp-minY) * yfactor - ypmin);
    return  ypmx - ypnt;
}

/**
 * Determines the some sizes, assigns the x and y values and the drawing
 * color.
 *
 * @param    xpoints   The x values
 * @param    ypoints   The y values
 * @param    col       The drawing color
 */
public void setupPlot(double[] xpoints, double ypoints[], Color col) {

    x = xpoints;    y = ypoints;    color = col;

    Insets insets = container.getInsets();
    int w = container.getWidth() - insets.left - insets.right - 40;
    int h = container.getHeight() - insets.top - insets.bottom - 10;

    xfactor = (0.9 * w) / (maxX - minX);
    yfactor = (0.9 * h) / (maxY - minY);

    xpmin = (int)(0.05 * w);    ypmx = (int)(0.05 * h);
    xpmx = w - xpmin;           ypmx = h - ypmx;
}

/**
 * Implementation of the drawing procedure. Default is white background
 * with a black rectangle (the frame of the viewing window). Subclasses should
 * call this method as <code>super.paintContent</code> and add the desired
 * graphics representing the chart.
 *
 * @param    g      The Graphics object to draw the chart into
 */
public void paintContent(Graphics g) {
    // erase background
    g.setColor(Color.white);
    g.fillRect(0, 0, container.getWidth(), container.getHeight());

    // draw bounding rectangle
    g.setColor(Color.black);
    g.drawRect(xpmin, ypmx, xpmx, ypmx);
}

} // eof CPlotAssistant

```

8 com.strato.lib.rule

8.1 CPalindromOfLength

```

package com.strato.lib.rule;

/*
 * @(#)CPalindromOfLength.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.*;
import java.util.*;

/**
 * Creates a string with contains a palindrom of given length
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.CRule
 * @since STRATO 1.0
 */

public final class CPalindromOfLength extends CRule {

    /** Constructor */
    public CPalindromOfLength() { super(); }

    /** Constructor */
    public CPalindromOfLength(String name) { super(name); }

    public CString createString(String params, int length, CAlphabet alphabet) {
        CString str = null;
        int len = 0;          // length of palindrom

        Vector v = this.extractParametersFromString(params);
        if (v.size() != 1) throw new RuntimeException("Invalid numbers of arguments");
        len = ((Integer)v.elementAt(0)).intValue();

        if (len > length) {
            return str;
        }
        else {
            str = new CString(length, alphabet);
            int pos = random.nextInt(length - len + 1);

            for(int i=0; i<len;i++) {
                byte r = (byte)(random.nextInt(alphabet.getSize()-1) + 1); // avoid JOKER -> +1
                str.setIndexAt(pos + i, r);           // from left
                str.setIndexAt(pos + len-1 - i, r); // from right
            }

            for(int i=0; i<length; i++) { // fill in remaining chars randomly
                if (str.getIndexAt(i) == CAlphabet.JOKER_INDEX) { // if not already set
                    byte r = (byte)(random.nextInt(alphabet.getSize()-1) + 1); // avoid JOKER -> +1
                    str.setIndexAt(i, r);
                }
            }
        }
        return str;
    }
} // eof CPalindromOfLength

```

8.2 CPatternAtLeastNTimes

```

package com.strato.lib.rule;

/*
 * @(#)CPatternAtLeastNTimes.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.*;
import java.util.*;

/**
 * Creates a string with contains a pattern at least (>=) a given number of times
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @see com.strato.framework.CRule
 * @since STRATO
 */
public final class CPatternAtLeastNTimes extends CRule {

    private BitSet bits = null;

    /** Constructor */
    public CPatternAtLeastNTimes() {
        super();
    }

    /** Constructor */
    public CPatternAtLeastNTimes(String name) {
        super(name);
    }

    public CString createString(String params, int length, CAlphabet alphabet) {
        CString str = null;
        String pattern = ""; // the substring to be placed
        int number = 0; // min number of substrings to be placed

        Vector v = this.extractParametersFromString(params);
        if (v.size() != 2) throw new RuntimeException("Invalid number of arguments");
        pattern = (String)v.elementAt(0);
        number = ((Integer)v.elementAt(1)).intValue();

        int charsNeeded = number * pattern.length();

        if (charsNeeded > length)
            return str;
        }
        else { // construct new String
            str = new CString(length, alphabet);

            bits = new BitSet(str.length());
            int patLen = pattern.length();
            for(int i=0; i<(str.length()-patLen+1); i++) { bits.clear(i); }
            int cnt = str.length() - patLen + 1; // still available starting positions

            // fill in the pattern (n times)
            while ((number > 0) && (cnt > 0)) {

                int nextRand = random.nextInt(cnt) + 1;
                int z = 0; int nextPos = -1;
                do {
                    nextPos++;
                    if (!bits.get(nextPos)) z++;
                } while (z < nextRand);
            }
        }
    }
}

```

```
// set the pattern from [nextPos..nextPos+patLen-1]
for (int i=0; i<patLen; i++) {
    int idx = nextPos + i;
    bits.set(idx);
    str.setCharAt(idx, pattern.charAt(i));
}
cnt -= patLen;

// mark additionally wasted starting positions
for(int i=1; i<patLen; i++) {
    int idx = ((nextPos - i) >= 0) ? nextPos-i : -1;
    if (idx != -1) {
        bits.set(idx);
        cnt--;
    }
    else break; // leave for-loop
}

number--;
}

for(int i=0; i<length; i++) { // fill in remaining chars randomly
    if (str.getIndexAt(i) == CAlphabet.JOKER_INDEX) { // if not already set
        int r = random.nextInt(alphabet.getSize()-1) + 1; // avoid JOKER -> +1
        str.setIndexAt(i, (byte)r);
    }
}

return str;
}
}

} //eof CPatternAtLeastNTimes
```

9 com.strato.tests

9.1 AllSubmassesFindingProblem

```

package com.strato.tests;

/*
 * @(#)AllSubmassesFindingProblem.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */
import com.strato.framework.*;
import com.strato.framework.test.*;

import java.util.*;

/**
 * ALL-SUBMASSES-FINDING-PROBLEM base class
 * <p>
 * This is the abstract superclass of all implementations of this special problem
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @since STRATO 1.0
 */
public abstract class AllSubmassesFindingProblem extends OneStringMassFindingProblem {

    /**
     * Constructor for this superclass of all ALL-SUBMASSES-FINDING-PROBLEM
     * solutions
     */
    public AllSubmassesFindingProblem(IAccess access) {
        super(access);
    }

    /* Algorithm: Does the submass exist? */
    public abstract void massExists();

    /** Algorithm: How many different submasses occur? */
    public abstract void massCount();

    /** Algorithm: List all occurring submasses! */
    public abstract void massList();

    public CAAlgorithmDescriptor[] getAlgorithmDescriptors() {
        CAAlgorithmDescriptor[] ad = super.getAlgorithmDescriptors();
        try {
            CAAlgorithmDescriptor algoCount =
                new CAAlgorithmDescriptor(this.getClass(), "massCount", "How many different submasses
exist?");
            CAAlgorithmDescriptor algoList =
                new CAAlgorithmDescriptor(this.getClass(), "massList", "List all occurring submasses
(sorted)?");
            CAAlgorithmDescriptor[] ret = {ad[0], algoCount, algoList};
            return ret;
        }
        catch (EConventionException ce) {
            throw new Error(ce.toString());
        }
    }
}

// eof AllSubmassesFindingProblem

```

9.2 BoyerMoore

```

package com.strato.tests;

/*
 * @(#)BoyerMoore.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.*;
import com.strato.framework.test.*;

import java.util.Vector;

/**
 * Exact Matching Algorithm (Boyer-Moore)
 * The implementation was translated from the "pseudocode" in the book:
 *   Algorithmen von Robert Sedgewick, 1992, Addison-Wesley
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @since STRATO 1.0
 */
public class BoyerMoore extends CExactMatchingTestInfo {

    /** preprocessed data: the delta values */
    protected int[] skip = null;

    /**
     * Constructor for Exact Matching using Boyer-Moore
     */
    public BoyerMoore(IAccess access) {
        super(access);
    }

    /**
     * preprocess the pattern
     */
    protected void preprocess() {
        skip = new int[inAlphabet.getSize()];
        int p = -1; int len = pattern.length();
        for(int i=0; i<skip.length; i++) {
            p = pattern.lastIndexOf(inAlphabet.getChar(i));
            skip[i] = (p != -1) ? len-(p+1) : len; // else init with M
        }
    }

    /**
     * The Boyer-Moore algorithm:
     *
     * @param i0          starting index for i
     * @param j0          starting index for j
     * @param doPreprocess  is preprocessing necessary
     */
    protected int bm(int i0, int j0, boolean doPreprocess) {
        int i = i0; int j = j0;
        int M = pattern.length(); int N = inString.length();

        if (doPreprocess) preprocess();

        if (pattern.length() > inString.length()) return Integer.MAX_VALUE;

        do {
            if (inString.charAt(i-1) == pattern.charAt(j-1)) {
                i--; j--;
            }
            else {
                int h = inString.indexOfAt(i-1);
                int a = M-j+1; int b = skip[inString.indexOfAt(i-1)];

```

```

        if (a>b) i+=a; else i+=b;
        j=M;
    }
} while((j>=1) && (i<=N));
return (i+1);
}

private int getFirstPos() {
    return bm(pattern.length(), pattern.length(), true);
}

private Vector getAllPos() {      // repeat the boyer-moore algorithm n-times
    Vector v = new Vector();

    int pos = -1;
    boolean firstLoop = true;
    int iInit = pattern.length();
    int jInit = pattern.length();

    do {
        if (iInit <= inString.length()) {
            pos = bm(iInit, jInit, firstLoop);
            firstLoop = false;
            if (pos<inString.length()) {      // one more occurence found
                v.add(new Integer(pos));
                iInit = pos+pattern.length();   // not optimal but easy to implement :-
                jInit = pattern.length();
            }
        } else { // force loop exit
            pos = inString.length();
        }
    } while(pos<inString.length());

    return v;
}

/* Algorithm: Does pattern exist? */
public void searchExists() {
    update();
    int pos = getFirstPos();
    boolean exists = (pos>inString.length()) ? false : true;

    // report
    outputs.Console().println("Pattern \\" +pattern+"\" "+((exists)?"" :"NOT ")+"found!");
    result[0] = new Boolean(exists);
}

/* Algorithm: How often does the pattern appear? */
public void searchCount() {
    update();
    Vector positions = getAllPos();
    int sum = positions.size();

    // report
    outputs.Console().println("Pattern \\" +pattern+"\" found "+sum+" times!");
    result[1] = new Integer(sum);
}

/* Algorithm: First occurrence of the pattern? */
public void searchFirst() {
    update();
    int pos = getFirstPos();

    // report
    if (pos>inString.length()) {
        pos = -1;
        outputs.Console().println("Pattern \\" +pattern+"\" not found!");
    } else
        outputs.Console().println("First occurrence of pattern \\" +pattern
                                +"\" found at position: "+pos);
    result[2] = new Integer(pos);
}

/* Algorithm: All occurrences of the pattern? */
public void searchAll() {

```

```
update();
Vector v = getAllPos();

// report
if (v.size() == 0)
    outputs.Console().println("Pattern \" " +pattern+" \" not found!");
else
    outputs.Console().println("Pattern \" " +pattern+" \" found at: "+v.toString());

result[3] = v;
}

/*****************/
/** Implementation of the ITestInfo interface **/
/*****************/

public CTestDescriptor getTestDescriptor() {
    CTestDescriptor test =
        new CTestDescriptor(BoyerMoore.class, "Exact Match (Boyer-Moore)");
    test.setShortDescription("Online Exact Matching algorithm using method of Boyer-Moore");
    return test;
}

***** all other descriptors are inherited from the superclasses *****

/*****************/
/** set some features provided by the super class (e.g. to be resetable) **/
/*****************/

public void reset() {
    // do here anything to reset your algorithm
    super.reset();
    this.result = new Object[4];
}

} // eof BoyerMoore
```

9.3 MultiplicityVectors

```

package com.strato.tests;

/*
 * @(#)MultiplicityVectors.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.*;
import com.strato.framework.test.*;

import java.util.*;

/**
 * Computes the multiplicity vectors (Parikh-vectors) of a string
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @since STRATO 1.0
 */

public class MultiplicityVectors extends Substrings {

    /** processed data: array of parikh-vectors */
    int[][] vectors;

    /**
     * Constructor for implementation gathering all parikh-vectors via substrings
     */
    public MultiplicityVectors(IAccess access) {
        super(access);
    }

    private int[] buildMV(CString str) {
        int[] res = new int[inAlphabet.getSize()];
        for(int i=0; i<str.length(); i++) {
            res[ str.getIndexAt(i) ]++;
        }
        return res;
    }

    protected void enumerateAllVectors() {
        long maxsize = countAllSubstrings();
        enumerateAllSubstrings(true); // calc all substrings (sorted)

        int[][] v = new int[(int)maxsize][];

        int len = inString.length();
        int i = substrings[substrings.length-1]; // start with beginning index
        int counter = 0;
        // generate substrings -> mv -> from the coded integer array
        outputs.setStatus("Generating vectors...");
        while(i != -1) {
            int cnt = substrings[i*2];
            for(int j=len-cnt; j<len; j++) {
                v[counter++] = buildMV(inString.substring(i, j+1));
            }
            i = substrings[i*2 + 1];
            outputs.setProgress((int)(counter*200/substrings.length));
        }

        outputs.setStatus("Sorting vectors...");
        // sort and copy to final destination
        Arrays.sort(v, new Comparator() {
            public int compare(Object o1, Object o2) {
                int[] a1 = ((int[])o1); int[] a2 = ((int[])o2);
                int len = a1.length;
                for(int i=0; i<len; i++) {

```

```

        if (a1[i] < a2[i]) return -1;
        else if (a1[i] > a2[i]) return +1;
    }
    return 0;
}

public boolean equals(Object obj) { return false; } // not used
});

outputs.setStatus("Eliminating duplicates...");
vectors = new int[v.length][]; int cnt = 0;
int[] last = v[cnt++]; vectors[0] = last;
for(int k=1; k<v.length; k++) {
    // compare v[k] to last (from right to left, cause of ordering) and take it
    // if it is a new vector
    boolean equal = true;
    for(int j=inAlphabet.getSize()-1; j>=0; j--) {
        if ((v[k][j] - last[j]) != 0) {
            equal = false; break;
        }
    }
    if (!equal) {
        vectors[cnt++] = v[k];
        last = v[k];
    }
    outputs.setProgress((int)(k*100/vectors.length));
}
for (int k=cnt; k<vectors.length; k++) {
    vectors[k] = null;
    outputs.setProgress((int)(k*100/vectors.length));
}

outputs.setProgress(100);
}

/** Algorithm: Count multiplicity vectors */
public void vectorCount() {
    update();

    enumerateAllVectors();
    int count = vectors.length;
    while(vectors[count-1] == null) { count --; }
    outputs.Console().println(count+" different multiplicity vectors found!");

    result[0] = new Long(count);
}

/** Algorithm: Enumerate multiplicity vectors */
public void vectorList() {
    update();

    enumerateAllVectors();
    outputs.Console().println("All multiplicity vectors (Parikh-Vectors):");
    for(int i=0; i<vectors.length; i++) {
        if (vectors[i] == null) break;
        outputs.Console().println(CStringUtils.arrayToString(vectors[i]));
    }

    result[1] = vectors;
}

/************************************************************/
/** Implementation of the ITestInfo interface **/
/************************************************************/

public CTestDescriptor getTestDescriptor() {
    return new CTestDescriptor(MultiplicityVectors.class, "Multiplicity Vectors (Parikh vectors)");
}

public CALgorithmDescriptor[] getAlgorithmDescriptors() {
    try {
        CALgorithmDescriptor algoCount =
            new CALgorithmDescriptor(MultiplicityVectors.class, "vectorCount", "Number of different multiplicity vectors?");
    }

    CALgorithmDescriptor algoList =

```

```
    new CAlgorithmDescriptor(MultiplicityVectors.class, "vectorList", "Enumerate all
multiplicity vectors!");

    CAlgorithmDescriptor[] ret = {algoCount, algoList};
    return ret;
}
catch (EConventionException ce) {
    throw new Error(ce.toString());
}
}

/************************************************************/
/** set some features provided by the super class (e.g. to be resetable) ***/
/************************************************************/

public void reset() {
    // do here anything to reset your algorithm
    super.reset();
    this.vectors = null;
    this.result = new Object[2];
}

} // eof MultiplicityVectors
```

9.4 OneStringMassFindingProblem

```

package com.strato.tests;

/*
 * @(#)OneStringMassFindingProblem.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */
import com.strato.framework.*;
import com.strato.framework.test.*;

import java.util.*;

/**
 * ONE-STRING-MASS-FINDING-PROBLEM base class
 * <p>
 * This is the abstract superclass of all implementations of this special problem
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @since STRATO 1.0
 */
public abstract class OneStringMassFindingProblem extends CSimpleTestInfo {

    /** parameter: the submass we search for */
    protected double m;

    /**
     * Constructor for this superclass of all ONE-STRING-MASS-FINDING-PROBLEM
     * solutions
     */
    public OneStringMassFindingProblem(IAccess access) {
        super(access);
    }

    /** Algorithm: Does the submass exist? */
    public abstract void massExists();

    /**
     * Implementation of the ITestInfo interface
     */
    public CParameterDescriptor[] getParameterDescriptors() {
        try {
            CParameterDescriptor mass =
                new CParameterDescriptor(this.getClass(), "m", Double.TYPE);
            mass.setDisplayName("submass M");
            mass.setShortDescription("The submass we search for");

            CParameterDescriptor[] ret = {mass};
            return ret;
        }
        catch (EConventionException ce) {
            throw new Error(ce.toString());
        }
    }

    public CALgorithmDescriptor[] getAlgorithmDescriptors() {
        try {
            CALgorithmDescriptor algoExists =
                new CALgorithmDescriptor(this.getClass(), "massExists", "Does given submass M
exist?");
            CALgorithmDescriptor[] ret = {algoExists};
            return ret;
        }
        catch (EConventionException ce) {
            throw new Error(ce.toString());
        }
    }
}

```

```
    }

    public void reset() {
        // do here anything to reset your algorithm
        super.reset();
        this.m = 1.0;
    }

    public double getM() {
        return this.m;
    }

    public void setM(double m) {
        this.m = m;
    }
}

// eof OneStringMassFindingProblem
```

9.5 OSMFPBinsearch

```

package com.strato.tests;

/*
 * @(#)OSMFPBinsearch.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */
import com.strato.framework.*;
import com.strato.framework.test.*;

import java.util.*;

/**
 * ONE-STRING-MASS-FINDING-PROBLEM (using BINSEARCH)
 *
 * The algorithm is explained in the paper
 *   Algorithmic Complexity of Protein Identification:
 *   Combinatorics of Weighted Strings
 *   by M. Cieliebak, Thomas Erlebach, Zsuzsanna Lipták, Jens Stoye, Emo Welzl
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @since STRATO 1.0
 */
public class OSMFPBinsearch extends AllSubmassesFindingProblem
    implements ITextResult {

    /** preprocessed data: the calculated submasses */
    protected double[] submasses;
    /** the string, which we calculate the submasses for */
    protected String curStr;

    /**
     * Constructor for BINSEARCH algorithm
     */
    public OSMFPBinsearch(IAccess access) {
        super(access);
    }

    /**
     * preprocess the input string -> calculate all occurring submasses
     */
    protected void preprocess() {
        Set set = Collections.synchronizedSet(new HashSet());
        curStr = inString.getAsString();
        int n = curStr.length();
        if (n>0) {
            // calc weights between positions {0,j} | j is element of [0..n-1]
            double sum = 0;
            double val[] = new double[n];
            for(int j=0; j<n; j++) {
                sum += inString.getWeightAt(j);
                val[j] = sum;
                set.add(new Double(sum));
            }
            // calc remaining values by lookup and use of precalculated values
            for(int l=1; l<n; l++) {
                outputs.setProgress((int)(l*100/n));
                for(int r=l; r<n; r++) {
                    set.add(new Double(val[r] - val[l-1]));
                }
            }
            Object[] masses = set.toArray();
            submasses = new double[masses.length];
            for(int i=0; i<submasses.length; i++) {
                submasses[i] = ((Double)masses[i]).doubleValue();
            }
        }
    }
}

```

```

    }

    // sort array
    Arrays.sort(submasses);
}
else {
    curStr = null;
    submasses = new double[0];
}
}

private boolean prepNeeded() {
    if (inString == null) return true;
    else return !inString.getAsString().equals(curStr);
}

/* Algorithm: Does the submass exist? */
public void massExists() {
    update();
    if (prepNeeded()) preprocess();

    int pos = Arrays.binarySearch(this.submasses, this.m);

    if (pos < 0)
        outputs.Console().println("Submass "+this.m+" not found!");
    else
        outputs.Console().println("Submass "+this.m+" found!");

    result[0] = new Boolean(pos >= 0);
}

/* Algorithm: How many different submasses occur? */
public void massCount() {
    update();
    if (prepNeeded()) preprocess();

    int count = this.submasses.length;

    outputs.Console().println("Number of submasses = "+count);

    result[1] = new Integer(count);
}

/* Algorithm: List all occurring submasses! */
public void massList() {
    update();
    if (prepNeeded()) preprocess();

    outputs.Console().println("Found submasses:");
    outputs.Console().println(CStringUtils.arrayToString(this.submasses));

    result[2] = this.submasses;
}

/*****************/
/** Implementation of the ITestInfo interface **/
/*****************/

public CTestDescriptor getTestDescriptor() {
    CTestDescriptor test =
        new CTestDescriptor(OSMFPBinsearch.class, "Binsearch {simple}");
    test.setShortDescription("Implements the BINSEARCH algorithm to solve "+
                           "the ONE-MASS-FINDING-PROBLEM");
    return test;
}

/*****************/
/** set some features provided by the super class (e.g. to be resetable) **/
/*****************/

public void reset() {
    // do here anything to reset your algorithm
    super.reset();
    submasses = null;
    curStr = null;
    this.result = new Object[3];
}

```

```

/*****************/
/** Interface ITextResult **/
/*****************/

public void write(javax.swing.text.JTextComponent tc, int algoID) throws EUpdateException {
    // constants
    String BEGIN = "<html><font color=blue size=+2>RESULT:</font><p>";
    String END = "</html>";

    String htmlres = null;

    try {
        switch(algoID) {
            case 0: // exists
                htmlres = BEGIN + "<pre>" +
                    ( (Boolean)result[algoID]).booleanValue() ?
                        "<font color=green size=+4>YES, submass "+m+" found!" :      // if found
                        "<font color=red size=+4>NO, submass "+m+" not found!" ) + // if not found
                    "</font></pre>" + END;
                break;

            case 1: // freq
                htmlres = BEGIN + "<ul>" +
                    "<li>There are " + result[algoID] + " different submasses found,</li>" +
                    "<li>in the String " + inString.getAsString() + "</li>" +
                    "</ul>" + END;
                break;

            case 2: // list
                double[] list = (double[])result[algoID];
                int row = list.length / 10;  if ((list.length % 10) > 0) row++;
                htmlres = BEGIN + "<table border=\"1\">";
                for (int i=0; i<row; i++) {
                    htmlres += "<tr>";
                    int col = ((i+1)*10 <= list.length) ? 10 : list.length % 10;
                    for(int j=0; j<col; j++)
                        htmlres += "<td>" + list[i*10+j] + "</td>";
                    htmlres += "</tr>";
                }
                htmlres += "</table>" + END;
                break;

            default: // -> should not occur
                // do nothing
                break;
        }

        tc.setText(htmlres); // display it
    }
    catch (Exception e) {
        throw new EUpdateException("");
    }
}
} // eof OSMFPBinsearch

```

9.6 OSMFPLinsearch

```

package com.strato.tests;

/*
 * @(#)OSMFPLinsearch.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.*;
import com.strato.framework.test.*;

import java.util.*;

/**
 * ONE-STRING-MASS-FINDING-PROBLEM (using LINSEARCH)
 *
 * The algorithm is explained in the paper
 *   Algorithmic Complexity of Protein Identification:
 *   Combinatorics of Weighted Strings
 *   by M. Cieliebak, Thomas Erlebach, Zsuzsanna Lipták, Jens Stoye, Emo Welzl
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @since STRATO 1.0
 */
public class OSMFPLinsearch extends OneStringMassFindingProblem {

    /** preprocessed data */
    // this test has none of them

    /**
     * Constructor for LINSEARCH algorithm
     */
    public OSMFPLinsearch(IAccess access) {
        super(access);
    }

    private int getFirstPos() {
        int l = 0;    int r = 0;
        double sum = inString.getWeightAt(0);
        int n = inString.length();

        while (r < n) {
            while(sum < this.m) {
                r++;
                if (r<n)
                    sum += inString.getWeightAt(r);
                else
                    break;
            }

            if (sum == this.m) return (l+1);

            while (sum > this.m) {
                sum -= inString.getWeightAt(l);
                l++;
            }
        }
        return -1; // submass not found
    }

    /* Algorithm: Does the submass exist? */
    public void massExists() {
        update();

        int pos = getFirstPos();

        if (pos == -1)

```

```

        outputs.Console().println("Submass "+this.m+" not found!");
    else
        outputs.Console().println("Submass "+this.m+" found!");

    result[0] = new Boolean(pos != -1);
}

/** Algorithm: Where does the mass m (the 1st time) occur? */
public void massFirst() {
    update();

    int pos = getFirstPos();

    if (pos == -1)
        outputs.Console().println("Submass "+this.m+" not found!");
    else
        outputs.Console().println("Submass "+this.m+" found at position "+pos);

    result[1] = new Integer(pos);
}

/************************************************************/
/** Implementation of the ITestInfo interface **/
/************************************************************/

public CTestDescriptor getTestDescriptor() {
    CTestDescriptor test =
        new CTestDescriptor(OSMFPLinsearch.class, "Linsearch {simple}");
    test.setShortDescription("Implements the LINSEARCH algorithm to solve "+
                            "the ONE-MASS-FINDING-PROBLEM");
    return test;
}

public CALgorithmDescriptor[] getAlgorithmDescriptors() {
    CALgorithmDescriptor[] ad = super.getAlgorithmDescriptors();
    try {
        CALgorithmDescriptor algoFirst =
            new CALgorithmDescriptor(OSMFPLinsearch.class, "massFirst", "Where does the mass m
(the 1st time) occur?");
        CALgorithmDescriptor[] ret = {ad[0], algoFirst};
        return ret;
    }
    catch (EConventionException ce) {
        throw new Error(ce.toString());
    }
}

/************************************************************/
/** set some features provided by the super class (e.g. to be resetable) **/
/************************************************************/

public void reset() {
    // do here anything to reset your algorithm
    super.reset();
    this.result = new Object[2];
}

} // eof OSMFPLinsearch

```

9.7 Statistics

```

package com.strato.tests;

/*
 * @(#)Statistics.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.*;
import com.strato.framework.test.*;
import com.strato.lib.output.*;

/**
 * Statistical evaluation of character frequencies in a given string.
 * <p>
 * It computes the character frequencies and supports several visualisation
 * functionality e.g. to present the frequencies as bar plot
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @since STRATO 1.0
 */
public class Statistics extends CSimpleTestInfo implements IGraphicResult {

    /** absolute frequencies of each character */
    protected int[] freqAbsolute = null;
    /** relative frequencies of each character */
    protected double[] freqRelative = null;

    /**
     * Constructor for statistical evaluations on the input string
     */
    public Statistics(IAccess access) {
        super(access);
    }

    /**
     * computes the frequencies
     *
     * @param isRelative if its true it computes the relative freq. too
     */
    protected void calcFrequencies(boolean isRelative) {
        freqAbsolute = new int[inAlphabet.getSize()];
        freqRelative = new double[inAlphabet.getSize()];

        for(int i=0; i<inString.length(); i++) {
            freqAbsolute[ inString.getIndexAt(i) ]++;
        }

        if (isRelative) {
            int sum = 0;
            for(int i=0; i<freqAbsolute.length; i++) {
                sum += freqAbsolute[i];
            }
            for(int i=0; i<freqRelative.length; i++) {
                freqRelative[i] = (double)freqAbsolute[i] / sum;
            }
        }
    }

    /**
     * Algorithm: Absolute frequency of the characters?
     */
    public void frequencyAbs() {
        update();
        calcFrequencies(false);
    }

    // report
}

```

```

        outputs.Console().println("Character set: "+inAlphabet.getAsString());
        outputs.Console().println("Absolute frequencies:");

        outputs.Console().println(CStringUtils.arrayToString(freqAbsolute));

        result[0] = freqAbsolute;
    }

    /**
     * Algorithm: Relative frequency of the characters?
     */
    public void freqencyRel() {
        update();
        calcFrequencies(true);

        // report
        outputs.Console().println("Character set: "+inAlphabet.getAsString());
        outputs.Console().println("Relative frequencies:");
        outputs.Console().println(CStringUtils.arrayToString(freqRelative));

        result[1] = freqRelative;
    }

/*****
/** Implementation of the ITestInfo interface */
****/

public CTestDescriptor getTestDescriptor() {
    CTestDescriptor test =
        new CTestDescriptor(Statistics.class, "Statistical evaluation");
    test.setShortDescription("Statistical evaluation of character frequencies in a given
string");
    return test;
}

public CALgorithmDescriptor[] getAlgorithmDescriptors() {
    try {
        CALgorithmDescriptor algoAbsFreq =
            new CALgorithmDescriptor(Statistics.class, "frequencyAbs", "Get a list of absolute
character frequencies");
        algoAbsFreq.setResultDescriptor(new CResultDescriptor(null, null));

        CALgorithmDescriptor algoRelFreq =
            new CALgorithmDescriptor(Statistics.class, "frequencyRel", "Get a list of relative
character frequencies");
        algoRelFreq.setResultDescriptor(new CResultDescriptor(null, null));

        CALgorithmDescriptor[] ret = {algoAbsFreq, algoRelFreq};
        return ret;
    }
    catch (EConventionException ce) {
        throw new Error(ce.toString());
    }
}

/*****
/** set some features provided by the super class (e.g. to be resetable) */
****/

public void reset() {
    // do here anything to reset your algorithm
    this.result = new Object[2];
}

/*****
/** interface IGraphicResult */
****/
private CPlotAssistant plotter = null;

public void setupPainting(java.awt.Container container, int algoID) throws EUpdateException
{
    try {

        // setup (x,y) pairs
        double[] x = new double[inAlphabet.getSize()];
        double[] y = new double[inAlphabet.getSize()];
        for(int i=0; i<x.length; i++) {

```

```
x[i] = (double)i;
switch (algoID) {
    case 0: // result is int[]
        y[i] = (double) ( ((int[])result[0])[i] ); break;
    case 1: // result is double[]
        y[i] = (double) ( ((double[])result[1])[i] ); break;
    default: // only to avoid traps
        y[i] = 0;
} // eof switch
}

// setup drawing aera (calculate viewport)
double maxY = 0.0;
for(int i=0; i<x.length; i++) { maxY = (y[i]>maxY) ? y[i] : maxY; }
switch (algoID) {
    case 0:
        maxY += 1.0;
        plotter = new CBarPlotAssistant(container);
        break;
    case 1:
        maxY *= 1.5; // there are all values < 1.0;
        plotter = new CLinePlotAssistant(container);
        break;
    default:
        maxY = 10.0;
}

// assign the calculated viewport and plot the values
plotter.setOutputBounds(0.0,0.0,x.length,maxY);
plotter.setupPlot(x, y, java.awt.Color.blue);

}
catch (Exception e) {
    throw new EUpdateException("");
}
}

public void paint(java.awt.Graphics g, int algoID) throws EUpdateException {
try {
    // delegate the painting job to the plotter
    plotter.paintContent(g);
}
catch (Exception e) {
    throw new EUpdateException("");
}
}

} // eof Statistics
```

9.8 Submasses

```

package com.strato.tests;

/*
 * @(#)Submasses.java 1.0 2002/03/03
 *
 * Title: STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright: Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company: ETH Zurich, Switzerland / Department of Computer Science
 * Homepages: http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.*;
import com.strato.framework.test.*;

import java.util.*;

/**
 * Computes the submasses of a string (making use of substrings array)
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @since STRATO 1.0
 */

public class Submasses extends AllSubmassesFindingProblem {

    /** preprocessed data: a sorted array of all occurring submasses */
    private double[] submasses;
    /** the string, which we calculate the submasses for */
    protected String curStr;

    /* ref to another test */
    private Substrings ssTest;

    /**
     * Constructor for the suffix tree based algorithm
     */
    public Submasses(IAccess access) {
        super(access);
    }

    private double calcWeight(CString str) {
        int sum = 0;    int size = inAlphabet.getSize();
        for(int j=0; j<str.length(); j++) {
            sum+= str.getWeightAt(j);
        }
        return sum;
    }

    private void preprocess() {
        curStr = inString.getAsString();
        ssTest = new Substrings(this.getAccess());
        int[] substrings = ssTest.enumerateAllSubstrings(true);    // delegate task

        int len = inString.length();
        int i = substrings[substrings.length-1]; // start with beginning index
        Set set = Collections.synchronizedSet(new HashSet());
        int counter = 0;
        // generate submasses from coded substrings array
        outputs.setStatus("Generating submasses...");
        while(i != -1) {
            int cnt = substrings[i*2];
            int lowerIntervalIdx = len-cnt;
            double currWeight = calcWeight(inString.substring(i, lowerIntervalIdx+1));
            set.add(new Double(currWeight));
            for(int j=lowerIntervalIdx+1; j<len; j++) { // calc remaining weights per leaf
                currWeight += inString.getWeightAt(j);
                set.add(new Double(currWeight));
            }
            i = substrings[i*2 + 1];
            counter++;
        }
    }
}

```

```

        outputs.setProgress((int)(counter*200/substrings.length));
    }
    // copy to array
    Object[] masses = set.toArray();
    submasses = new double[masses.length];
    for(int k=0; k<submasses.length; k++) {
        submasses[k] = ((Double)masses[k]).doubleValue();
    }
    outputs.setStatus("Sorting submasses...");
    Arrays.sort(submasses);
}

private boolean prepNeeded() {
    if (inString == null) return true;
    else return !inString.getAsString().equals(curStr);
}

/* Algorithm: Does the submass exist? */
public void massExists() {
    update();
    if (prepNeeded()) preprocess();

    int pos = Arrays.binarySearch(this.submasses, this.m);

    if (pos < 0)
        outputs.Console().println("Submass "+this.m+" not found!");
    else
        outputs.Console().println("Submass "+this.m+" found!");

    result[0] = new Boolean(pos >= 0);
}

/* Algorithm: How many different submasses occur? */
public void massCount() {
    update();
    if (prepNeeded()) preprocess();

    int count = this.submasses.length;

    outputs.Console().println("Number of submasses = "+count);

    result[1] = new Integer(count);
}

/* Algorithm: List all occurring submasses! */
public void massList() {
    update();
    if (prepNeeded()) preprocess();

    outputs.Console().println("Found submasses:");
    outputs.Console().println(CStringUtils.arrayToString(this.submasses));

    result[2] = this.submasses;
}

/******************
/** Implementation of the ITestInfo interface **/
/******************/

public CTestDescriptor getTestDescriptor() {
    return new CTestDescriptor(Submasses.class, "Submasses (Suffix-Trees)");
}

/******************
/** set some features provided by the super class (e.g. to be resetable) **/
/******************/

public void reset() {
    // do here anything to reset your algorithm
    super.reset();
    this.submasses = null;
    this.curStr = null;
    this.ssTest = null;
    this.result = new Object[3];
}

} // eof Submasses

```

9.9 Substrings

```

package com.strato.tests;

/*
 * @(#)Substrings.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.*;
import com.strato.framework.test.*;
import com.strato.lib.ds.*;

/**
 * Computes the substrings of a given input string using suffix trees
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @since STRATO 1.0
 */
public class Substrings extends CSimpleTestInfo {

    /** preprocessed data: the built suffix tree */
    private CSuffixTree stree;
    /** the string, which we calculate the submasses for */
    private String curStr;

    /** processed data: the calculated substrings in a special notation */
    protected int[] substrings;

    /**
     * Constructor for the suffix tree implementation gathering all substrings
     */
    public Substrings(IAccess access) {
        super(access);
    }

    /**
     * preprocess the input string -> calculate the suffix tree
     */
    private final void preprocess() {
        substrings = null;
        curStr = inString.getAsString();
        stree = new CSuffixTree(inString);
    }

    /**
     * check if we have to call method <code>preprocess</code>
     */
    private final boolean prepNeeded() {
        if (inString == null) return true;
        else return !inString.getAsString().equals(curStr);
    }

    /* package view -> can be used by other tests without subclassing this class */
    long countAllSubstrings() {
        if (prepNeeded()) preprocess();
        return stree.countSubstrings();
    }

    /* package view -> can be used by other tests without subclassing this class */
    int[] enumerateAllSubstrings(boolean sorted) {
        if (prepNeeded()) preprocess();
        if (sorted)
            substrings = stree.enumerateSortedSubstrings();
        else
            substrings = stree.enumerateSubstrings();
        return substrings;
    }
}

```

```

}

/** Algorithm: Count all substrings */
public void substringCount() {
    update();

    long count = countAllSubstrings();
    outputs.Console().println(count+" different substrings found!");
    result[0] = new Long(count);
}

/** Algorithm: Enumerate all substrings */
public void substringList() {
    update();

    enumerateAllSubstrings(false);
    outputs.Console().println("The following substrings found (unsorted):");

    int len = inString.length();
    // generate substrings from the coded integer array
    for (int i=0; i<substrings.length; i++) {
        int cnt = substrings[i];
        for(int j=len-cnt; j<len; j++) {
            outputs.Console().print(inString.substring(i, j+1) + " ; ");
        }
    }
    outputs.Console().println();
    result[2] = null;
    result[1] = substrings;
}

/** Algorithm: Enumerate all substrings in lexicalic order */
public void substringSorted() {
    update();

    enumerateAllSubstrings(true);
    outputs.Console().println("The following substrings found (alphabetically sorted):");

    int len = inString.length();
    int i = substrings[substrings.length-1]; // start with beginning index
    // generate substrings from the coded integer array
    while(i != -1) {
        int cnt = substrings[i*2];
        for(int j=len-cnt; j<len; j++) {
            outputs.Console().print(inString.substring(i, j+1) + " ; ");
        }
        i = substrings[i*2 + 1];
    }
    outputs.Console().println();
    result[1] = null;
    result[2] = substrings;
}

/*****************/
/** Implementation of the ITestInfo interface */
/*****************/

public CTestDescriptor getTestDescriptor() {
    return new CTestDescriptor(Substring.class, "Substring exploration");
}

public CAAlgorithmDescriptor[] getAlgorithmDescriptors() {
    try {
        CAAlgorithmDescriptor algoCount =
            new CAAlgorithmDescriptor(Substring.class, "substringCount", "Number of different
substrings?");

        CAAlgorithmDescriptor algoList =
            new CAAlgorithmDescriptor(Substring.class, "substringList", "Enumerate all substrings
(unsorted)!");

        CAAlgorithmDescriptor algoSorted =
            new CAAlgorithmDescriptor(Substring.class, "substringSorted", "Enumerate all substrings
(sorted)!");
    }

    CAAlgorithmDescriptor[] ret = {algoCount, algoList, algoSorted};
    return ret;
}

```

```
        catch (EConventionException ce) {
            throw new Error(ce.toString());
        }
    }

/************************************************************/
/** set some features provided by the super class (e.g. to be resetable) ***/
/************************************************************/

public void reset() {
    // do here anything to reset your algorithm
    super.reset();
    this.stree = null;
    this.curStr = null;
    this.substrings = null;
    this.result = new Object[3];
}

} // eof Substrings
```

9.10 SuffixTreeSearch

```

package com.strato.tests;

/*
 * @(#)SuffixTreeSearch.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.*;
import com.strato.framework.test.*;
import com.strato.lib.ds.*;

/**
 * Exact Matching Algorithm (Suffix Trees)
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @since STRATO 1.0
 */
public class SuffixTreeSearch extends CExactMatchingTestInfo {

    /** preprocessed data: the built suffix tree **/
    protected CSuffixTree stree;

    /** the string, which we construct the suffix tree for */
    private String curStr;

    /**
     * Constructor for Exact Matching using Suffix Trees
     */
    public SuffixTreeSearch(IAccess access) {
        super(access);
    }

    /**
     * preprocess the input string -> calculate the suffix tree
     */
    private final void preprocess() {
        curStr = inString.getAsString();
        stree = new CSuffixTree(inString);
    }

    /**
     * check if we have to call method <code>preprocess</code>
     */
    private final boolean prepNeeded() {
        if (inString == null) return true;
        else return !inString.getAsString().equals(curStr);
    }

    private Result[] search() {
        if (prepNeeded()) preprocess();
        Result[] r = stree.searchPattern(new CString(this.pattern, inAlphabet));
        return r;
    }

    private int[] getResultAsSortedArray(Result[] res) {
        int[] vec = new int[res.length];
        for(int i=0; i<res.length; i++) {
            vec[i] = res[i].suffix + 1;      // we count from 1 to n
        }
        java.util.Arrays.sort(vec);
        return vec;
    }

    /* Algorithm: Does pattern exist? */
    public void searchExists() {

```

```

update();
int count = ((Result[])search()).length;
boolean exists = (count > 0);

// report
outputs.Console().println("Pattern \\" +pattern+"\\ "+((exists)?"":"NOT ")+"found!");
result[0] = new Boolean(exists);
}

/* Algorithm: How often does the pattern appear? */
public void searchCount() {
    update();
    int count = ((Result[])search()).length;

    // report
    outputs.Console().println("Pattern \\" +pattern+"\\ found "+count+" times!");
    result[1] = new Integer(count);
}

/* Algorithm: First occurrence of the pattern? */
public void searchFirst() {
    update();
    Result[] r = search();

    int pos = -1;
    int[] vec = getResultAsSortedArray(r);

    // report
    if (vec.length == 0) {
        outputs.Console().println("Pattern \\" +pattern+"\\ not found!");
    }
    else {
        pos = vec[0];
        outputs.Console().println("First occurrence of pattern \\" +pattern
                                +"\\ found at position: "+pos);
    }

    result[2] = new Integer(pos);
}

/* Algorithm: All occurrences of the pattern? */
public void searchAll() {
    update();
    Result[] r = search();

    //Generate array
    int[] vec = getResultAsSortedArray(r);

    // report
    if (vec.length == 0)
        outputs.Console().println("Pattern \\" +pattern+"\\ not found!");
    else {
        outputs.Console().println("Pattern \\" +pattern+"\\ found at: "+
                                CStringUtil.arrayToString(vec));
    }

    result[3] = vec;
}

/******************
/** Implementation of the ITestInfo interface **/
/******************/

public CTestDescriptor getTestDescriptor() {
    CTestDescriptor test =
        new CTestDescriptor(SuffixTreeSearch.class, "Exact Match (Suffix Tree)");
    test.setShortDescription("Exact Matching algorithm using a Suffix Tree");
    return test;
}

/******************
/** set some features provided by the super class (e.g. to be resetable) **/
/******************/

public void reset() {
    // do here anything to reset your algorithm
    super.reset();
}

```

```
    this.curStr = "dummy";
    this.result = new Object[4];
}

} // SuffixTreeSearch
```

9.11 WSIntervalWeight

```

package com.strato.tests;

/*
 * @(#)WSIntervalWeight.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.*;
import com.strato.framework.test.*;

/**
 * Computes the weight of a string in a specified index intervall
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @since STRATO 1.0
 */
public class WSIntervalWeight extends CSimpleTestInfo implements ITextResult {

    /** parameter: the index we start from */
    private int lowerIndexBound;
    /** parameter: the index we end at */
    private int upperIndexBound;

    /** preprocessed data */
    // this test has none of them

    /**
     * Constructor for the weight calculation algorithm
     */
    public WSIntervalWeight(IAccess access) {
        super(access);
    }

    private double calcWeight(int left, int right) { // [0..N-1]
        int len = inString.length();
        double sum = 0.0;
        if (len > 0) { // calc String has chars
            // check range
            if (left<0) left = 0;
            if (right>len) right = len;

            for(int i=left; i<right; i++) {
                sum+= inString.getWeightAt(i);
            }
        }
        return sum;
    }

    /**
     * Algorithm: Is overall weight in intervall [lowerIndexBound, upperIndexBound]?
     */
    public void between() {
        update();
        double w = calcWeight(lowerIndexBound-1, upperIndexBound);

        // report
        outputs.Console().println("Weight of interval ["+lowerIndexBound+","+upperIndexBound+"] is
= "+w);

        result[0] = new Double(w);
    }

    /**
     * Algorithm: Return overall weight!
     */
    public void overall() {

```

```

update();
double w = calcWeight(0,inString.length());

// report
outputs.Console().println("Overall weight = "+w);

result[1] = new Double(w);
}

/*****************/
/** Implementation of the ITestInfo interface **/
/*****************/

public CTestDescriptor getTestDescriptor() {
    CTestDescriptor test =
        new CTestDescriptor(WSIntervalWeight.class, "Interval Weight");
    test.setShortDescription("Computes the weight of a string in a specified "+
                           "index intervall");
    return test;
}

public CALgorithmDescriptor[] getAlgorithmDescriptors() {
    try {
        CALgorithmDescriptor algoBetween =
            new CALgorithmDescriptor(WSIntervalWeight.class, "between", "The weight of intervall
[

```

```
public void setLowerIndexBound(int lowerIndexBound) {
    this.lowerIndexBound = lowerIndexBound;
}

public int getUpperIndexBound() {
    return this.upperIndexBound;
}

public void setUpperIndexBound(int upperIndexBound) {
    this.upperIndexBound = upperIndexBound;
}

/** Interface ITextResult */
/***********************/

public void write(javax.swing.text.JTextComponent tc, int algoID) throws EUpdateException {
    String HTML_TEXT_EXAMPLE = "<html>In case you thought that text outputs had to  
be<p>boring," +
        "one line descriptions, the <font color=blue size=+2>Results</font>" +
        " can shatter your illusions.<p>In STRATO, they can use HTML to <ul><li>" +
        " Have Lists<li><b>Bold</b> text<li><em>emphasized</em> text<li>text with" +
        " <font color=red>Color</font><li>text in different <font size=+3>sizes</font><li>" +
        "and <font face=AvantGarde>Fonts</font></ul>Oh, and they can be multi-line, too.</html>";

    try {
        tc.setText("<html><font color=blue size=+2>RESULT:</font><p><pre>" +
            + result[algoID] + "</pre></html>");
    }
    catch (Exception e) {
        throw new EUpdateException("");
    }
}
} // eof WSIntervalWeight
```

9.12 WSSubmasses

```

package com.strato.tests;

/*
 * @(#)WSSubmasses.java    1.0    2002/03/03
 *
 * Title:      STRING-Analysis-TOols (STRATO)
 * Description: Diploma Thesis: Structural Properties of Strings
 * Copyright:   Copyright (c) 2001, 2002 by Daniel Emmenegger
 * Company:    ETH Zurich, Switzerland / Department of Computer Science
 * Homepages:  http://www.ethz.ch , http://www.ti.inf.ethz.ch/pw
 *
 * For further information see LICENCE.TXT
 */

import com.strato.framework.*;
import com.strato.framework.test.*;

import java.util.*;

/**
 * Computes the submasses of a string
 *
 * @author Daniel Emmenegger
 * @version 1.0 2002/03/03
 * @since STRATO 1.0
 */

public class WSSubmasses extends MultiplicityVectors {

    /** parameter: the submass we search for */
    protected double m;

    /** preprocessed data: a sorted array of all occurring submasses */
    private double[] submasses;
    /** the string, which we calculate the submasses for */
    protected String curStr;

    /**
     * Constructor for the suffix tree based algorithm
     */
    public WSSubmasses(IAccess access) {
        super(access);
    }

    private double calcWeight(int[] v) {
        int sum = 0;    int size = inAlphabet.getSize();
        for(int j=0; j<size; j++) {
            sum+= v[j] * inAlphabet.getWeightOf(j);
        }
        return sum;
    }

    private void preprocess() {
        enumerateAllVectors();    // call superclass

        Set set = Collections.synchronizedSet(new HashSet());
        for(int i=0; i<vectors.length; i++) {
            if (vectors[i] == null) break;
            set.add( new Double(calcWeight(vectors[i])) );
        }

        Object[] masses = set.toArray();
        submasses = new double[masses.length];
        for(int i=0; i<submasses.length; i++) {
            submasses[i] = ((Double)masses[i]).doubleValue();
        }

        // sort array
        Arrays.sort(submasses);
    }

    private boolean prepNeeded() {
        if (inString == null) return true;
}

```

```

        else return !inString.getAsString().equals(curStr);
    }

    /**
     * Algorithm: Does the submass exist?
     */
    public void exists() {
        update();
        if (prepNeeded()) preprocess();

        int pos = Arrays.binarySearch(this.submasses, this.m);

        if (pos < 0)
            outputs.Console().println("Submass "+this.m+" not found!");
        else
            outputs.Console().println("Submass "+this.m+" found!");

        result[0] = new Boolean(pos >= 0);
    }

    /**
     * Algorithm: How many different submasses occur?
     */
    public void number() {
        update();
        if (prepNeeded()) preprocess();

        int count = this.submasses.length;

        outputs.Console().println("Number of submasses = "+count);

        result[1] = new Integer(count);
    }

    /**
     * Algorithm: List all occurring submasses!
     */
    public void list() {
        update();
        if (prepNeeded()) preprocess();

        outputs.Console().println("Found submasses:");
        outputs.Console().println(CStringUtil.arrayToString(this.submasses));

        result[2] = this.submasses;
    }

/*****
** Implementation of the ITestInfo interface **
*****/



public CTestDescriptor getTestDescriptor() {
    return new CTestDescriptor(WSSubmasses.class, "Submasses (Suffix-Tree)");
}

public CALgorithmDescriptor[] getAlgorithmDescriptors() {
    try {
        CALgorithmDescriptor algoExists =
            new CALgorithmDescriptor(WSSubmasses.class, "exists", "Does given mass m exist?");

        CALgorithmDescriptor algoNumber =
            new CALgorithmDescriptor(WSSubmasses.class, "number", "How many different submasses
exist?");

        CALgorithmDescriptor algoList =
            new CALgorithmDescriptor(WSSubmasses.class, "list", "List all occurring submasses?");

        CALgorithmDescriptor[] ret = {algoExists, algoNumber, algoList};
        return ret;
    }
    catch (EConventionException ce) {
        throw new Error(ce.toString());
    }
}

public CParameterDescriptor[] getParameterDescriptors() {
    try {

```

```
CParameterDescriptor mass =
    new CParameterDescriptor(WSSubmasses.class, "m", Double.TYPE);
mass.setDisplayName("submass M");
mass.setShortDescription("The submass we search for");

CParameterDescriptor[] ret = {mass};
return ret;

}
catch (EConventionException ce) {
    throw new Error(ce.toString());
}
}

/************************************************************/
/** set some features provided by the super class (e.g. to be resetable) ***/
/************************************************************/

public void reset() {
    super.reset();
    // do here anything to reset your algorithm
    this.m = 1.0;

    this.submasses = null;
    this.curStr = null;

    this.result = new Object[3];
}

/************************************************************/
/** do not forget: the getter and setter Methods for the parameters ***/
/************************************************************/

public double getM() {
    return this.m;
}

public void setM(double m) {
    this.m = m;
}

} // eof WSSubmasses
```

Übersicht: Sourcefiles von STRATO Version 1.0

Package com.strato.	Klasse	# Zeilen
cui	CInputStream	92
	CStratoBatch	244
	CStratoCmd	247
framework	CAlphabet	397
	CRule	222
	CString	415
	CStringStreamable	179
	CStringUtil	117
	ENoSuchCharacterException	44
	<i>IAccess</i>	53
	<i>IInput</i>	43
	<i>IOOutput</i>	50
	<i>IPersistence</i>	62
	<i>IStreamable</i>	43
	<i>IString</i>	41
framework.gui	JDrawPanel	135
	JWritePane	114
framework.run	CAbstractCase	62
	CRunner	203
	CSTMICase	37
	CSTSICase	72
framework.test	CAlgorithmDescriptor	82
	CConvention	219
	CCtrlCenter	97
	CExactMatchingTestInfo	106
	CGenericDescriptor	111
	CParameterDescriptor	169
	CReferenceDescriptor	71
	CResultDescriptor	68
	CSimpleTestInfo	165
	CTestDescriptor	86

	EConventionException	47
	EInvalidUserInputException	43
	EUpdateException	44
	<i>IGraphicResult</i>	50
	<i>IResult</i>	28
	<i>ITestInfo</i>	68
	<i>ITestResult</i>	38
	TDescriptorShortcuts	57
gui	Strato	620
	StratoActionLib	214
	StratoApplet	56
	StratoConfig	66
	StratolconLib	173
	StratoInputPanel	629
	StratoLanguage	86
	StratoMenuBar	612
	StratoModule	155
	StratoOutputPanel	264
	StratoRunnable	45
	StratoToolBar	99
	StratoSwitchToAction	46
gui.input	CAbstractInput	86
	CFileInput	66
	CFileManager	162
	CRandomInput	114
	CRegularInput	111
	CRuleInput	114
	CTextInput	166
	<i>IInputViews</i>	47
	TAlphabetModel	79
gui.misc	CodeViewer	"417"
	CTextPrintable	"179"
	GenericFileFilter	"104"
	JExtFileChooser	"1951"
gui.module	Tests	743

gui.output	CAbstractOutput	157
	CConsoleOutput	104
	CGraphicalOutput	83
	CTextualOutput	102
gui.themes	AquaTheme	"58"
	CharcoalTheme	"72"
	ContrastTheme	"103"
	EmeraldTheme	"57"
	RubyTheme	"57"
lib	CRandom	208
lib.alphabet	CAlphaNumeric	46
	CBinary	47
	CDigits	47
	CDna	47
	CHexadecimal	55
	CLowerCase	51
	CNotQuiteASCII	67
	CProteine	56
	CUpperCase	51
	CUserDefined	80
lib.ds	CSuffixTree	218
	Result	"35"
	SuffixArray	"322"
	SuffixTree	"237"
	Watch	"61"
lib.input.javacc	ParseException	191
	PseudoRegular	681
	PseudoRegularConstants	66
	PseudoRegularTokenManager	491
	SimpleCharStream	279
	Token	73
	TokenMgrError	133

lib.output	CBarPlotAssistant	48
	CLinePlotAssistant	54
	CPlotAssistant	134
lib.rule	CPalindromOfLength	73
	CPatternAtLeastNTimes	107
tests	AllSubmassesFindingProblem	64
	BoyerMoore	189
	EndlessLoop	141
	MultiplicityVectors	172
	OneStringMassFindingProblem	91
	OSMFPBinsearch	207
	OSMFPLinsearch	133
	Statistics	196
	Submasses	154
	Substrings	168
	SuffixTreeSearch	157
	WSIntervalWeight	182
	WSSubmasses	193

Package	Klasse	# Zeilen
framework	XP_CAlphabet	87
	XP_CRule	143
	XP_CString	88
	XP_CStringUtils	79
framework.test	XP_CConvention	74
	XP_CSimpleTestInfo	95
	XP_CTestInfo	188

TOTAL: 15'724