Aufgabe:

Die Aufgabe besteht aus 2 Teilen:
Zum Ersten sollen wir ein Programm schreiben, das jedes Wort in einem Satz „twistet“. Zweitens sollen wir ein Programm schreiben, welches einen Satz wiederum „enttwistet“.

Ein Wort ist dann getwistet, wenn alle Buchstaben, bis auf den ersten und den letzten, eines Wortes gemischt sind. So ist „unbeaufsichtigt“ getwistet „utagcuiebhsnfit“.

Lösungsidee:

Einen Satz twisten:

Zuerst teilen wir den Satz in einzelne Wörter auf. Danach erstellen für jedes Wort einen Array mit allen Buchstaben, bis auf den ersten und letzten, und vermischen diese. Danach ersetzen wir die Buchstaben, in den noch nicht getwisteten Wörtern, mit den Vermischten aus dem Array.

Operation	Wort String	Buchstaben Array
Buchstaben in den Array	unbeaufsichtigt	nbeaufsichtig
Array vermischen	unbeaufsichtigt	tagcuiebhsnfi
Buchstaben im String ersetzen	utagcuiebhsnfit	tagcuiebhsnfi

Nachdem alle Wörter getwistet wurden, fügen wir sie wieder zusammen und geben den Satz aus.

Einen Satz enttwisten:

Beim enttwisten wird es komplizierter. Wie auch beim twisten teilen wir den Satz als erstes in Wörter auf, mit denen wir weiterarbeiten können. Auch lesen wir die Wörterliste mit allen deutschen Wörtern ein.

Jetzt gehen wir für jedes Wort die gesamte Wörterliste durch und vergleichen alle Attribute, die beim twisten unverändert bleiben:

1. die ersten Buchstaben
2. die Letzten Buchstaben
3. die Länge der Wörter
4. alle Buchstaben der Wörter
    - Buchstaben alphabetisch sortiert, und verglichen

Sobald bei einem Wort in der Wörterliste alle Bedingungen erfüllt sind, ersetzen wir das getwistete Wort mit diesem.

Nachdem alle Wörter enttwistet wurden, fügen wir sie wieder zusammen und geben den Satz aus.

Umsetzung

Im Vorfeld müssen wir, für beide Aufgaben, den eingegebenen Satz in einzelne Wörter unterteilen.

Zuerst erstellen wir eine ArrayList aus Strings in der wir die Wörter speichern. Danach gehen wir mit einer while Schleife, Char für Char, den String sentenceString durch und erstellen jedes Mal ein neues ‚Wort' (=String in der ArrayList). Wenn der betrachtete Char von einem Buchstaben zu einem Zeichen oder andersrum wechselt, so werden Wörter und Zeichenfolgen einzeln abgespeichert.

`"Der Mann sagte ‚Hallo!' und lief"`	`["Der", " ", "Mann", " ", "sagte", " ‚", "Hallo", "!' ", "und", " ", "lief"]`

Twisten:

Wir unterteilen den Satz wie oben beschrieben.

Für jedes Wort überprüfen wir zuerst ob der Char ein Buchstabe ist. Wenn nicht, dann handelt es sich um eine Zeichenfolge und wir überspringen dieses Wort. Wenn doch, dann erstellen wir zwei Char Arrays, einen mit allen Chars aus dem Wort (fullUntwistedWord) und einen mit nur den mittleren Chars (charsToTwist). Daraufhin mischen wir charsToTwist mithilfe des Durstenfeld-Shuffles. Die mittleren Chars, also nicht der erste und letzte Buchstabe, von fullUntwistedWord werden auf den Wert von charsToTwist gesetzt. Und wir erstellen einen String aus den Werten von fullUntwistedWord.

Wir fügen alle Strings zusammen und geben den getwisteten Satz aus.

Enttwisten:

Wie auch beim Twisten teilen wir den Satz in Wörter auf. Daraufhin gehen wir mit foreach Schleife die Wörter durch, enttwisten sie und fügen sie direkt zum output String hinzu.

Für das Enttwisten lesen wir auch noch die Liste aller deutschen Wörter ein und speichern sie in einer ArrayList (wordlist). Zusätzlich gibt es die Option eine englische Wörterliste einzulesen.

Bei der Methode zum Wörter enttwisten überprüfen wir wieder, ob der erste Char ein Buchstabe ist. Danach prüfen wir, ob die Länge des Wortes größer als 3 ist. Wenn eine dieser Abfragen false liefert geben wir das Wort so wie es ist wieder zurück.

Wir erstellen einen Char Array (twistedWord_Sorted) mit allen Buchstaben des getwisteten Wortes und sortieren ihn alphabetisch. Dieser wird jedoch erst an späterer Stelle benötigt.

Danach gehen wir mit einer foreach Schleife die wordlist durch und vergleichen:

1. die Länge der Wörter
2. die ersten Buchstaben
    - vor Vergleichen setzen wir alle auf lowercase
3. die letzten Buchstaben

Wenn von einem Wort alle Bedingungen erfüllt sind, dann wird ein Char Array mit allen Buchstaben des Wortes erstellt. Er wird dann alphabetisch sortiert und mit twistedWord_Sorted verglichen.

Falls sie gleich sind, passen wir die Groß-/Kleinschreibung an und geben das Wort aus wordlist zurück. Falls kein Wort gefunden werden sollte, wird die getwistete Version zurückgegeben.

Quellcode:

Untwist

```
public class Untwist {
  public static ArrayList<String> wordlist;

  /**
   * Woerter werden getrennt und in der Arraylist hinzugefuegt
   *
   * @param sentence a group of words or a sentence
   * @return
   */
  public static ArrayList<String> stringToArrayList(String sentenceString) {

    ArrayList<String> sentenceList = new ArrayList<String>();
    String word = "";
    int i = 0;
    int lastLetter = sentenceString.length() - 1;

    while (i <= lastLetter) {
      boolean isLetter = Character.isLetter(sentenceString.charAt(i));

      while (Character.isLetter(sentenceString.charAt(i)) == isLetter && i <= lastLetter) {

        word += sentenceString.charAt(i);
        i++;
        if (i > lastLetter)
          break;
      }

      sentenceList.add(word);
      word = "";
    }
    return sentenceList;
  }



  /**
   * untwisting the sentence
   *
   * @param sentence Arraylist created before, from sentence
   * @return
   */
  public static String untwistSentence(ArrayList<String> sentence) {
    String output = "";
    for (String twistedWord : sentence) {
      output += untwistWord(twistedWord, wordlist) + "";
    }
    return output;
  }

  /**
   * untwisting the word
   *
   * @param twistedWord twistedWord from untwistSentence
   * @param wordlist
   * @return
   */
  public static String untwistWord(String twistedWord, ArrayList<String> wordlist) {

    // first and last letter stay and the one in the middle has no to change
    // with
    boolean isWord = Character.isLetter(twistedWord.charAt(0));

    if (isWord) {
```

```
    boolean wordIsTwistable = twistedWord.length() > 3;

    if (wordIsTwistable) {

      String twistedWord_Lowered =
          twistedWord.toLowerCase();
      char firstLetter_TwistedWord =
          twistedWord_Lowered.charAt(0);
      char lastLetter_TwistedWord =
          twistedWord_Lowered.charAt(twistedWord.length() - 1);
      String twistedWord_Sorted = SortString(twistedWord_Lowered);

      // go through each word in the list
      for (String currWord : wordlist) {

        boolean lengthOfWordsAreEqual =
            currWord.length() == twistedWord.length();

        if (lengthOfWordsAreEqual) {

          String currWord_Lowered = currWord.toLowerCase();
          char firstLetter_CurrWord = currWord_Lowered.charAt(0);
          boolean firstLetters_AreEqual =
              firstLetter_TwistedWord == firstLetter_CurrWord;

          if (firstLetters_AreEqual) {

            char lastLetter_CurrWord =
                currWord_Lowered.charAt(currWord.length() - 1);
            boolean lastLetters_AreEqual =
                lastLetter_CurrWord == lastLetter_TwistedWord;

            if (lastLetters_AreEqual) {
              // now it is likely to have the correct word

              // we now sort all letters alphabetical and
              // compare the letters with each other
              String currWord_Sorted =
                  SortString(currWord_Lowered);
              boolean wordsAreEqual =
                  currWord_Sorted.equals(twistedWord_Sorted);

              // not perfect but close enough
              if (wordsAreEqual) {
                currWord = adaptCase(currWord, twistedWord);
                return currWord;
              } // if
            } // if
          } // if
        } // if
      } // for
    } // if
  } // if

  // when the word can't be untwisted, it stays how it is
  return twistedWord;
}

/**
 * adapting the lower and upper case from twisted word to untwisted word
 *
 * @param currWord    the untwisted word currently
 * @param twistedWord the twisted word from currWord
 * @return
 */
```

```
  private static String adaptCase(String currWord, String twistedWord) {
    StringBuilder wordBuilder = new StringBuilder(currWord);
    wordBuilder.setCharAt(0, twistedWord.charAt(0));
    currWord = wordBuilder.toString();
    return currWord;
  }

  /**
   * sorting the String
   *
   * @param unsortedString the String which have to be sorted
   * @return
   */
  private static String SortString(String unsortedString) {
    char[] charArray = unsortedString.toCharArray();
    java.util.Arrays.sort(charArray);
    String sortedString = new String(charArray);
    return sortedString;
  }
}
```

Twist

```
public class Twist {
  /**
   * @param Normal sentence
   * @return Twisted sentence
   */
  private static String twistSentence(ArrayList<String> sentence) {
    String output = "";
    for (String untwistedWord : sentence) {
      output += twistWord(untwistedWord) + "";
    }
    return output;
  }
  /**
   * @param String with multiple words and -or character.
   * @return Words and characters saved separately in an ArrayList.
   */
  public static ArrayList<String> stringToArrayList(String sentence) {
    ArrayList<String> wordlist = new ArrayList<String>();
    String word = "";
    int i = 0;
    int lastLetter = sentence.length() - 1;
    while (i <= lastLetter) {
      if (Character.isLetter(sentence.charAt(i))) {
        while (Character.isLetter(sentence.charAt(i)) && i <= lastLetter) {
          word += sentence.charAt(i);
          i++;
          if (i > lastLetter)
            break;
        }
        wordlist.add(word);
        word = "";
      } else {
        while (!Character.isLetter(sentence.charAt(i)) && i <= lastLetter) {
          word += sentence.charAt(i);
          i++;
          if (i > lastLetter)
            break;
        }
        wordlist.add(word);
        word = "";
      }
    }
    return wordlist;
  }
```

```java
  /**
   * @param word untwisted word
   * @return twisted word
   */
  private static String twistWord(String word) {
    if (word.length() > 3) {
      if (Character.isLetter(word.charAt(0))) {
        char[] fullUntwistedWord = word.toCharArray();
        char[] CharsToTwist = new char[fullUntwistedWord.length - 2];
        for (int i = 0; i < CharsToTwist.length; i++) {
          CharsToTwist[i] = fullUntwistedWord[i + 1];
        }
        char[] twistedChars = DurstenfeldShuffle(CharsToTwist);
        for (int i = 0; i < twistedChars.length; i++) {
          fullUntwistedWord[i + 1] = twistedChars[i];
        }
        return new String(fullUntwistedWord);
      }
    }
    return word;

  }

  /**
   * @param charArray
   * @return shuffled charArray
   */
  private static char[] DurstenfeldShuffle(char[] ar) {
    Random rnd = new Random();
    for (int i = ar.length - 1; i > 0; i--) {
      int index = rnd.nextInt(i + 1);
      // Simple swap
      char a = ar[index];
      ar[index] = ar[i];
      ar[i] = a;
    }
    return ar;
  }
}
```