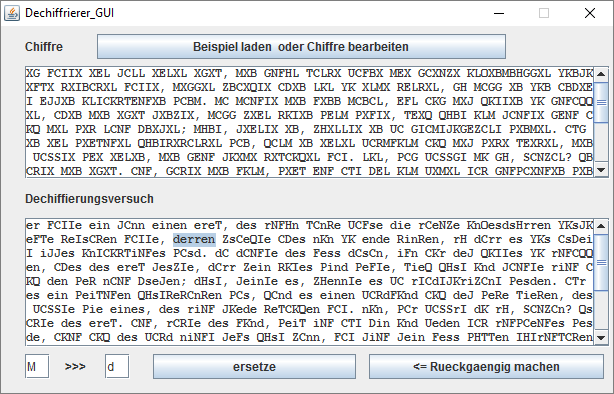
**Projekt Undo (Klasse Stack)**

**Beispiel: Handdechiffrierer**

Bei dem vorliegenden Text handelt es sich um eine Chiffre, die dadurch entstanden ist, dass in einem deutschsprachigen Text Buchstaben durch andere Buchstaben getauscht worden sind. Leerzeichen und Satzzeichen sind nicht chiffriert worden.



Es ist eine Software „Handdechiffrierer“ entwickelt worden, die bei der Dechiffrierung des Textes behilflich ist. Es werden die Ersetzungen, die ausprobiert werden sollen, auf die vorliegende Chiffre angewandt und im unteren Textbereich angezeigt.

Es kommt schon einmal vor, dass man beim Ersetzen eine falsche Idee verfolgt, die man dan wieder rückgängig machen möchte.

Handhabung: Man kann im oberen Textfeld eine Chiffre eingeben oder kopieren. Bei Knopfdruck auf „Beispiel laden oder Chiffre bearbeiten“ wird die Chiffre ins untere Fenster kopiert und die Ersetzungsversuche können gemacht werden. Falls die Fenster leer sind, wird eine Beispielchiffre geladen.

Bei dem o.a. Beispiel sind die folgenden Ersetzungen vorgenommern worden: (X,e), (L,n), (E,i), (G,r), (B,s), (M,d). An dem markierten Wort kann man erkennen, dass die vorgenommenen Ersetzungen nicht zum Ziel führen können. Die Ersetzung (G,r) ist offensichtlich falsch. Die Software ermöglicht es, die vorgenommen Ersetzungen bis zu dieser Stelle zurück zu nehmen.

Um die Ersetzungen zu verwalten, benötigt man eine Datenstruktur, die den zuletzt getätigten Eintrag als erstes wieder zurück gibt, also eine Organisation nach dem „Last In First Out“-Prinzip.

Das LIFO-Prinzip ähnelt einem vertikalen Bücherstapel. Elemente werden in genau der entgegengesetzten Reihenfolge abgerufen, in der sie zuvor abgelegt wurden, das heißt, das erste („unterste“) Element wird als letztes abgerufen. Entsprechend wird das letzte („oberste“) Element zuerst abgerufen.

Die Operation, bei der ein neues Objekt auf den Stapel gelegt wird, nennt man „Push“. Bei der Operation „Pop“ wird das oberste Objekt (Top of Stack) gelesen und gleichzeitig gelöscht. Sollte das oberste Objekt nur gelesen werden und nicht gelöscht, so wird eine Operation namens „Top“ benutzt.

Die Bezeichnungen Push und Pop entstanden aus der Ähnlichkeit zu einem Stapel von Tabletts in einer Cafeteria: Wird ein Tablett auf den Stapel gestellt, wird der Stapel hinuntergedrückt (engl. to push ‚drücken‘); wird ein Tablett vom Stapel entfernt, „poppt“ der Stapel nach oben.

Alle Programmiersprachen benutzen LiFo-Speicher ([Stapelspeicher](https://de.wikipedia.org/wiki/Stapelspeicher)) für interne Zwecke, die [Stapelorientierten Sprachen](https://de.wikipedia.org/wiki/Stapelspeicher#Stapelorientierte_Sprachen) auch für die Operationen mit Daten.

(Wikipedia, <https://de.wikipedia.org/wiki/Last_In_%E2%80%93_First_Out>, 20.07.2015)

Klasse Stack

150720_Stack

Die generische Klasse **Stack<ContentType>**

Objekte der generischen Klasse Stack (Keller, Stapel) verwalten beliebige Objekte vom Typ

**ContentType** nach dem Last-In-First-Out-Prinzip, d.h., das zuletzt abgelegte Objekt wird als

erstes wieder entnommen. Alle Methoden haben eine konstante Laufzeit, unabhängig von

der Anzahl der verwalteten Objekte.

Dokumentation der generischen Klasse **Stack<ContentType>**

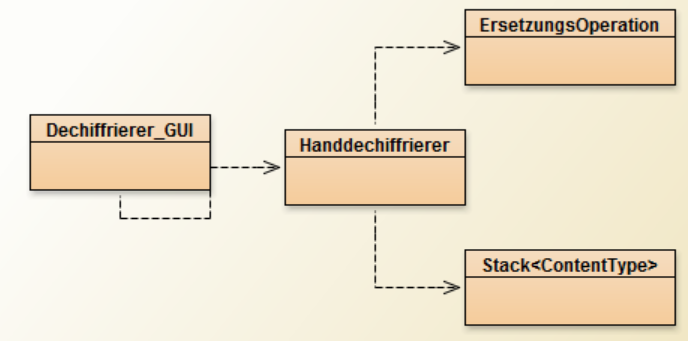
|  |  |
| --- | --- |
| **Konstruktor** | **Stack<ContentType>()** |
|  | Ein leerer Stapel wird erzeugt. Objekte, die in diesem Stapel  verwaltet werden, müssen vom Typ **ContentType** sein. |
| **Anfrage** | **boolean isEmpty()** |
|  | Die Anfrage liefert den Wert **true**, wenn der Stapel keine Objekte  enthält, sonst liefert sie den Wert **false**. |
| **Auftrag** | **void push(ContentType pContent)** |
|  | Das Objekt **pContent** wird oben auf den Stapel gelegt. Falls  **pContent** gleich **null** ist, bleibt der Stapel unverändert. |
| **Auftrag** | **void pop()** |
|  | Das zuletzt eingefügte Objekt wird von dem Stapel entfernt. Falls der  Stapel leer ist, bleibt er unverändert. |
| **Anfrage** | **ContentType top()** |
|  | Die Anfrage liefert das oberste Stapelobjekt. Der Stapel bleibt  unverändert. Falls der Stapel leer ist, wird **null** zurückgegeben. |

Lösungsskizze:

Implementationsdiagramm



Darstellung in BlueJ



Implementation der Klasse **Handdechiffrierer**:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27**  **28**  **29**  **30**  **31**  **32** | **public class Handdechiffrierer {**  **private String chiffre, neuerText;**  **private Stack<ErsetzungsOperation> undoStapel;**  **public Handdechiffrierer() {**  **chiffre = "";**  **undoStapel = new Stack<ErsetzungsOperation>();**  **}**  **public void setzeChiffre(String pText) {**  **chiffre = pText;**  **neuerText = chiffre;**  **}**  **public String gibChiffre() {**  **return chiffre;**  **}**  **public String gibDecodiertenText() {**  **return neuerText;**  **}**  **public void ersetze(String pAlt, String pNeu) {**  **neuerText = neuerText.replace(pAlt,pNeu);**  **ErsetzungsOperation undo =**  **new ErsetzungsOperation(pAlt.charAt(0), pNeu.charAt(0));**  **undoStapel.push(undo);**  **}**  **public void rueckgaengig() {**  **if (!undoStapel.isEmpty()) {**  **ErsetzungsOperation undo = undoStapel.top();**  **String alt = "" + undo.gibOriginalZeichen();**  **String neu = "" + undo.gibErsetztDurchZeichen();**  **neuerText = neuerText.replace(neu, alt);**  **undoStapel.pop();**  **}**  **}**  **}** |