Prof. Dr. Ralf Hinze
Markus Heinrich, M.Sc.

**TU Kaiserslautern**
**Fachbereich Informatik**
**AG Programmiersprachen**

# Miniprojekt 1: Programmierpraktikum 2021

Ausgabe: 11. Mai 2021
Abgabe: 26. Mai 2021, 15 Uhr

## GitLab Team Repositories

Für die Miniprojekte und Projekte nutzen wir das Versionsverwaltungssystem Git. Unser GitLab Server ist dabei eine zentrale Kopie des Repositories und bietet zudem eine Weboberfläche mit weiteren Funktionen.

Für jedes Abgabeteam haben wir ein eigenes GitLab Projekt erstellt. Darin befindet sich ein Ordner `MP1`, der die Vorlage für das erste Miniprojekt enthält. Clonen (`git clone`) Sie sich das Repository Ihres Teams und öffnen Sie dann den `MP1` Ordner in IntelliJ IDEA. Darin enthalten ist ein Gradle Projekt. IntelliJ IDEA lädt automatisch die definierten Abhängigkeiten herunter, wenn Sie das Projekt zum ersten Mal öffnen.

Ihre Lösungen müssen Sie in das lokale Repository committen (`git commit`) und dann auf den GitLab Server übertragen (`git push`). Diese beiden Git Funktionen stehen auch direkt in IntelliJ IDEA bereit. Sehen Sie sich dazu die weiteren Videos an. Sprechen Sie sich zur Bearbeitung der Aufgaben mit Ihren Abgabepartner*innen ab, damit Sie unnötige Konflikte vermeiden. In den Vorlesungen zu Git lernen Sie, wie Sie dennoch aufgetretene Konflikte beheben können.

## Projektabnahmen und Betreuung

Bei der Durchführung der Miniprojekte und Projekte werden Sie von einer Tutorin/einem Tutor betreut. Die/der Tutor*in steht Ihnen für Rückfragen zur Verfügung und wird nach der Abgabe eine Projektabnahme durchführen, bei der jedes Teammitglied Teile der Abgabe erklären muss. So möchten wir sicherstellen, dass sich alle an der Abgabe beteiligt haben.

# Die Schnittstelle Map

Im ersten Miniprojekt wollen wir eine `Map` Schnittstelle auf verschiedene Arten implementieren. Eine `Map` assoziiert Schlüssel mit Werten, sie ist also eine endliche Abbildung. Zu jedem eingetragen Schlüssel ist ein Wert gespeichert. Wenn ein neuer Wert mit demselben Schlüssel eingetragen wird, dann überschreibt dieser den bisher gespeicherten Wert.

```java
public interface Map<K, V> {
    V get(K key);
    void put(K key, V value);
    void remove(K key);
    int size();
    void keys(K[] array);
}
```

Die `Map` ist dabei generisch bzgl. der Typen für Schlüssel und Werte. `K` ist der Typ für die Schlüssel, `V` der Typ für die Werte. Eine `Map<String, Integer>` weist also jedem eingetragenen `String` eine ganze Zahl zu und eine `Map<Integer, String>` weist Zahlen einen `String` zu.

Die Methode `V get(K key)` gibt den zum Schlüssel `key` gespeicherten Wert zurück, oder **`null`**, wenn kein Wert zu diesem Schlüssel gespeichert ist. Die Methode `void put(K key, V value)` speichert für den Schlüssel `key` den Wert `value`. Falls bereits ein Eintrag für `key` existiert hat, so wird dieser überschrieben. Die Methode `void remove(K key)` löscht den Eintrag für Schlüssel `key` falls dieser existiert, ansonsten bleibt die `Map` unverändert. Die Methode `int size()` gibt die Anzahl der in der `Map` gespeicherten Einträge zurück.

Die Methode `void keys(K[] array)` bietet eine Möglichkeit, alle in der `Map` gespeicherten Schlüssel abzufragen: Sie erwartet als Eingabe ein Array ausreichender Größe und speichert darin die in der `Map` eingetragenen Schlüssel ab. Die Reihenfolge der Schlüssel spielt dabei keine Rolle, jeder Schlüssel muss aber exakt ein Mal ins Array geschrieben werden. Falls das gegebene Array **`null`** ist oder wenn es zu klein ist, soll eine `IllegalArgumentException` geworfen werden[1]. Wenn das gegebene Array zu groß ist, werden die Indizes `0` bis `size() - 1` befüllt, der Rest des Arrays bleibt unverändert. Der Grund, warum die Methode *nicht* die Signatur `K[] keys()` hat, ist dass das Erstellen eines Arrays mit einem generischen Basistypen in Java nicht möglich ist. Das liegt an der sogenannten *Type Erasure*, die dazu führt, dass der Typparameter `K` zur Laufzeit nicht mehr bekannt ist.

Unsere Schnittstelle ähnelt stark der `java.util.Map`, bietet aber einen geringeren Funktionsumfang, sodass sie einfacher zu implementieren ist.

**Zur Lösung der Aufgaben aus diesem Miniprojekt dürfen Sie die Java Standardbibliothek *nicht* verwenden!** Sie müssen also alle benötigten Klassen und Methoden selbst implementieren. Dabei sollen Ihre Kompetenzen im Umgang mit den grundlegenden Sprachfeatures gefestigt werden. In der Praxis würde man niemals eine eigene `Map` Schnittstelle definieren, sondern die `java.util.Map` mitsamt ihren Implementierungen aus der Standardbibliothek benutzen.

[1] `throw new IllegalArgumentException();`

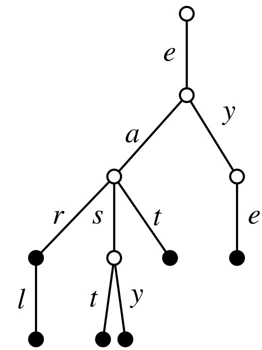
## Aufgabe 1 ListMap: Implementierung mit einer verketteten Liste

Implementieren Sie eine Klasse **public class** ListMap<K, V> **implements** Map<K, V>. Die Einträge der Map sollen in einer verketteten Liste verwaltet werden. Sie brauchen also eine separate Klasse für Listenelemente, in der Schlüssel, Wert und das nächste Listenelement gespeichert sind. Die einzelnen Methoden aus der Map Schnittstelle müssen diese Liste dann durchlaufen.

## Aufgabe 2 TrieMap: Implementierung mit einem Trie

Implementieren Sie die Klasse **public class** TrieMap<V> **implements** Map<String, V>. Wie Sie sehen, handelt es sich um eine spezielle Map, die nur Strings als Schlüssel zulässt. Die Einträge der Map sollen in einer Suchbaumstruktur verwaltet werden. Sie brauchen also eine separate Klasse, um die Baumknoten zu repräsentieren.

Ein Trie hat nicht wie ein Binärbaum genau zwei Kindelemente, sondern eine variable Anzahl. Man macht sich zunutze, dass die Schlüssel aus Strings, also aus einer Sequenz von Zeichen bestehen. Die Verbindung zwischen einem Knoten und seinen Kindelementen wird jeweils mit einem einzelnen Zeichen versehen. Die nebenstehende Abbildung zeigt ein Beispiel für solch einen Trie. Knoten, die Werte enthalten, sind schwarz markiert. Der Trie enthält also die Schlüssel "ear", "earl", "east", "easy", "eat", "eye" mit entsprechenden Werten, die im Endknoten des jeweiligen Pfades hinterlegt sind. Die durch die Kanten dargestellte Zuordnung kann mit einer „normalen“ Map realisiert werden. Verwenden Sie dazu ListMap<Character, V> von Teilaufgabe a, bzw. wenn Sie mit dieser Probleme haben, kopieren Sie die LibraryMap vom Ordner test und verwenden Sie diese.




Beim Entfernen eines Schlüssels kann entweder nur der zugehörige Wert entfernt (auf **null** gesetzt) werden oder, sofern der Knoten keine weiteren Kindknoten besitzt, kann dieser ganz aus der Baumstruktur entfernt werden (sauberer, aber aufwändiger). Dies lässt sich am vorher genannten Beispiel illustrieren: Beim Löschen von "earl" kann entweder der Wert des Knotens entfernt werden, auf den die Kante 'l' verweist (um einen weißen Knoten zu erhalten). Da der letzte Knoten des Pfades keine weiteren Kinder enthält, kann er aber auch komplett entfernt werden.

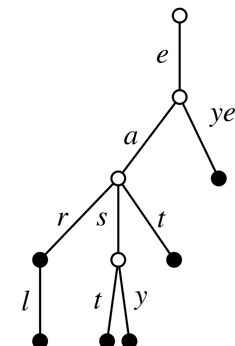
Mit der Operation get stellen wir nur Existenzanfragen an den Trie, also ob ein bestimmter Schlüssel enthalten ist (und welcher Wert ihm zugeordnet ist). Tries eigenen sich jedoch auch dazu, Präfixanfragen effizient zu beantworten, also welche Schlüssel der Trie enthält, die mit einer vorgegebenen Zeichenkette beginnen. Ein Anwendungsbeispiel dafür wäre z. B. Autovervollständigung (s. Aufgabe 5).

## Aufgabe 3 PatriciaTrieMap: Implementierung mit einem Patricia Trie

*Freiwillige Zusatzaufgabe.*

Implementieren Sie die Klasse **public class** PatriciaTrieMap<V> **implements** Map<String, V>.

Den Trie aus der vorherigen Teilaufgabe können wir hinsichtlich Platzbedarf optimieren. Anstelle einzelner Zeichen, können wir auch Strings in den Kanten speichern. Pfade, in denen Knoten ohne Wert und mit nur einem Kindknoten auftreten, lassen sich verkürzen. Im Beispiel der vorherigen Aufgabe ließe sich der zum Schlüssel "eye" gehörige rechte Teilbaum verkürzen zu einer Kante "e" (unverändert) und "ye". Die put Operation zum Einfügen wird damit komplizierter: Möchten wir zum Beispiel den Schlüssel "eyry" einfügen, muss die Kante "ye" gesplittet werden in eine Kante "y", die auf einen Knoten zeigt, der zwei ausgehende Kanten "e" und "ry" hat.




Entsprechend ergibt sich bei der remove Operation die Herausforderung bei Knoten mit genau einem Kindelement, dass die eingehende Kante des Knotens (in dem der Wert gelöscht wurde) mit der ausgehenden Kante verschmolzen werden sollte.

## Aufgabe 4 Häufigkeitsanalyse

Wir möchten die `Map` verwenden, um zu zählen, wie oft einzelne Wörter in der Bibel vorkommen. Wir haben Ihnen die Methode `static Iterable<String> Util.getBibleWords()` bereitgestellt, die Wort für Wort den Text aus der mitgelieferten Bibel Datei liefert. Das `Iterable` können Sie mit einer **for** Schleife iterieren:

```
for (String word : Util.getBibleWords()) {
    System.out.println(word);
}
```

Verwenden Sie eine derartige **for** Schleife, um in der Klasse `BibleAnalyzer` zunächst die Klassenmethode `static void countWords(Map<String, Integer> counts)` zu implementieren. Sie soll zählen, wie oft jedes Wort in der Bibel vorkommt. Die Häufigkeiten sollen in die übergebene `Map` eingetragen werden. Beachten Sie dabei, dass der Typ `Integer` ein Referenztyp ist, der neben den Werten des Basistyps `int` auch `null` sein kann. Berechnungen wie `null + 1` oder die Umwandlung eines `Integers` mit dem Wert `null` in einen `int` würden eine `NullPointerException` werfen. Sie müssen also bei Verwendung der `get` Methode prüfen, ob das Ergebnis `null` ist, bevor Sie weiter damit arbeiten.

Implementieren Sie dann die `main` Methode in dieser Klasse:

1. Sie soll eine `Map<String, Integer>` instanziieren und die zuvor implementierte Methode `countWords` damit aufrufen. Welche der Klassen Sie als konkrete `Map` Implementierung nutzen ist dabei zunächst egal.
2. Speichern Sie die Wörter (ohne Duplikate) in einem Array vom Typ `String[]`.
3. Sortieren Sie das Array aufsteigend nach Häufigkeit der einzelnen Wörter. Implementieren Sie dazu in der Klassenmethode `static void sort(String[] words, Map<String, Integer> counts)` ein geeignetes Sortierverfahren, welches das übergebene Array "in-place" sortiert. Wenn Sie Hilfsmethoden benötigen ergänzen Sie einfach weitere Klassenmethoden, diese können dann auch `private` sein.

   Rufen Sie die `sort` Methode dann von der `main` Methode aus auf.

   Gerne können Sie den Code für den Sortieralgorithmus aus der GdP Vorlesung, aus anderen Vorlesungen oder sonstigen Quellen übernehmen. Geben Sie in einem Kommentar an, um welches Sortierverfahren es sich handelt und ergänzen Sie einen Link zur Quelle. Sie dürfen jedoch *nicht* die Sortierfunktionalitäten der Java Standardbibliothek aufrufen.
4. Abschließend sollen alle Wörter auf die Konsole geschrieben werden. Eines pro Zeile, und zwar zuerst die Häufigkeit gefolgt von einem Leerzeichen und dem Wort. Beispiel:

   ```
   42 Engel
   123 Gott
   254 Amen
   ```

Die Antworten auf die folgenden Fragen müssen Sie nicht abgeben; sie werden in der Abnahme diskutiert:

Welches sind die 10 häufigsten Wörter in der Bibel? Welches ist das häufigste Nomen?

Untersuchen Sie, wie sich die unterschiedlichen `Map` Implementierungen auf die Ausführungszeit auswirken. Nennen Sie Gründe für die Unterschiede.

# Aufgabe 5 Autovervollständigung

*Freiwillige Zusatzaufgabe.*

Wie bereits erwähnt, können Tries zur effizienten Implementierung von Autovervollständigung eingesetzt werden. Dazu lesen wir zunächst ähnlich wie in der vorherigen Aufgabe die Wörter der Bibel ein. Jedoch interessiert uns hier nicht, wie oft ein Wort vorkommt. Implementieren Sie in der Klasse `BibleAutocomplete` die Klassenmethode `static void getWords(TrieMap<Boolean> words)`, die für jedes in der Bibel vorkommende Wort einen Eintrag in der TrieMap `words` mit dem Wert `true` ablegt. Erweitern Sie Ihre Trie Implementierung um eine Methode `public void prefixKeys(String prefix, String[] array)`, die alle Schlüssel, die mit `prefix` beginnen, in das übergebene Array schreibt. Implementieren Sie dann damit in der Klasse `BibleAutocomplete` die Klassenmethode `public static void printAutocompletes(TrieMap<Boolean> words, String prefix)`, welche alle mit `prefix` beginnenden Schlüssel ausgibt.

In der `main` Methode soll ein Präfix von der Konsole eingelesen werden und alle Wörter der Bibel ausgegeben werden, welche mit diesem Präfix beginnen. Beispiel:

```
Enter␣word:␣Engel↩
Engelzungen
Engeln
Engelbrot
Engels
```

Im Beispiel ist die Ausgabe des Programms `blau` und die Eingabe `rot` markiert. Leerzeichen sind durch das Symbol ␣ dargestellt, Zeilenumbrüche durch ↩.