Kontrollfragen A

1. Kennen Sie weitere Beispiele aus dem Alltag, welche das Wechselspiel zwischen Organisation/Ablage und Handling illustrieren?
   Bibliothek: Bücherregal (nach Thema, Titel, Autor)

2. Die früher behandelten Suchalgorithmen machten auch unterschiedlich von Datenstrukturen (Variablen und Arrays) Gebrauch. Welcher Algorithmus war diesbezüglich der einfachste, welcher der anspruchsvollste?
   Einfach: einfache Suche
   Anspruchsvoll: optimal Missmatch

3. Was ist charakteristisch für ein Set, eine List oder eine Map?
   Set: Eindeutigkeit (ein Objekt kann nicht doppelt eingetragen werden) (Key)
   List: beliebige Grösse (Value)
   Map: jeweils Objekt-Paare (Key / Value)

Kontrollfragen B

1. Erklären Sie mit Ihren Worten, was es heisst, wenn für den Aufwand für das Einfügen eines Elementes in eine Datenstruktur O(1) gilt.
   Linear, egal wie gross die Liste ist der Aufwand ist immer gleich gross.

2. Handelt es sich bei einer ArrayList um eine statische oder dynamische Datenstruktur?
   ArrayList hat keine fest grösse => dynamische Datenstruktur

3. Mit welcher Zugriffsart haben wir es typisch bei Listen zu tun?
   Indirekt (zugriff z.B. via Iterator)

4. Welche Operation ist bei einem unsortierten Array besonders effizient?
   Einfügen: O(1)

5. Welche Operation ist bei einem sortierten Array besonders effizient?
   *Suchen O(n * log(n))*

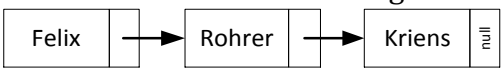
6. Was spricht alles für eine sortierte Liste?
   Dynamische Datenstruktur (beliebige Grösse)
   Die Liste ist sortiert (suche)
   Nachfolge und Vorgänger Element sind bekannt

Kontrollfragen C

1. Deklarieren und instanzieren Sie eine Liste namens `list`.
   Fügen Sie nun Ihren Vornamen, dann Ihren Familiennamen und weiter noch Ihren Wohnort als Strings in `list` ein.

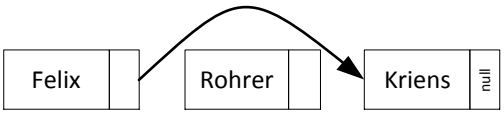
```
import java.util.LinkedList;
private LinkedList<String> list = new LinkedList <String>();
list.add("Felix");
list.add("Rohrer");
list.add("Kriens");
```

2. Illustrieren Sie Ihre `list` möglichst anschaulich mit einer Zeichnung (nicht UML-like!).



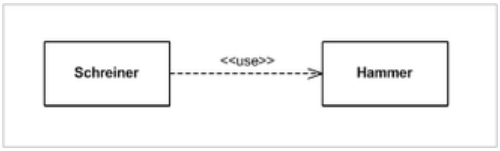
3. Entfernen Sie nun den Familiennamen aus der Liste, d.h. arbeiten Sie gedanklich die Methode `remove()` Anweisung für Anweisung ab. Führen Sie dabei Ihre Zeichnung nach.

```
list.remove(1)
```

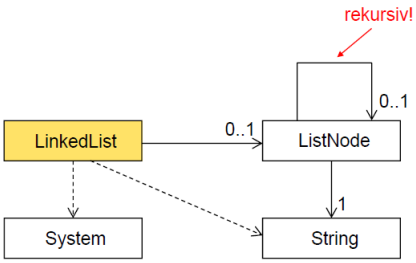


4. Wir setzen NEU anstelle von `String` überall `Object` als Klassentyp ein. Was erreichen wir damit?
   Die Liste kann nun nicht nur Strings aufnehmen, sondern sämtliche Objekte (generisch).

5. Im Klassendiagramm haben wir es mit `use` und `has_a` Beziehungen zu tun. Ist Ihnen der Unterschied klar?
   use: *Die* Verwendungsbeziehung *(Usage / <<use>>) gibt an, dass das abhängige Element das unabhängige Element benutzt.*



   has_a: *Variablen, LinkedList „enthält" ListNode.*



Kontrollfragen D

1. Was für Vorteile bringen uns Generics bei Datenstrukturen?
   Wiederverwendbarkeit
   Klasse nur 1-mal implementieren
   Typsicherheit ist garantiert (Compiler verifiziert Typen)

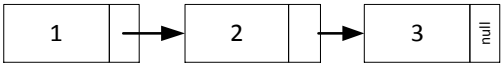
2. Eine Liste ist eine rekursive Datenstruktur. Weshalb?
   Ein Element zeigt jeweils auf das nächste Element.

3. Die Methode `isFound()` ruft `equals()` auf.
   Funktioniert dies wirklich problemlos bei verschiedenen Klassentypen <T> ?
   Nein, dies funktioniert nur bei vorhandenen Typen aus der Klassenbibliothek.
   Bei eigenen Datentypen muss .equals() spezifisch implementiert werden.

4. Machen Sie für die Methode `main()` einen Walkthrough.
   Gehen Sie insbesondere das Abarbeiten von `print()` genau durch. Machen Sie ein paar Illustrationen dazu.

```
public static void main(String[] args)
{
  LinkedList<Integer> list = new LinkedList<Integer>();
  list.insert(new Integer(1));
  list.insert(new Integer(2));
  list.insert(new Integer(3));
  list.print();
  list.remove(new Integer(2));
  list.print();
}
```

```
list.insert(new Integer(1));
list.insert(new Integer(2));
list.insert(new Integer(3));
```




```
list.remove(new Integer(2));
```