Hochschule Karlsruhe
University of Applied Sciences

Fakultät für **Informatik**
und Wirtschaftsinformatik

Prof. Dr. Carsten Sinz
Lehrgebiet Software-Entwicklung

Übungsblatt 5 zur Vorlesung Programmieren (fällig bis 07.12.25)

Aufgabe 1 (Abstrakter Datentyp Liste):

Implementieren Sie den Datentyp "Abstrakte Liste" für Listen ganzer Zahlen (`int`). Hier die relevanten Methodenrümpfe (das "Interface"):

- `public void addFirst(int x)`
- `public Integer getFirst()`
- `public boolean dropFirst()`
- `public void addLast(int x)`
- `public Integer getLast()`
- `public boolean dropLast()`
- `public void remove(int x)`
- `public boolean contains(int x)`
- `public int size()`
- `public boolean isEmpty()`

Die Spezifikation der Methoden soll so wie in der Vorlesung sein. Für die in der Vorlesung nicht vorgestellten Methoden `getFirst`, `dropFirst`, `getLast`, `dropLast` soll folgendes gelten: `getFirst` liefert das erste Element der Liste zurück, falls es ein solches gibt, ansonsten `null`, `getLast` soll analog funktionieren. `dropFirst` entfernt das erste Element aus der Liste, `dropLast` das letzte. Wenn ein Element aus der Liste entfernt wurde, wird `true` zurückgegeben, ansonsten (leere Liste) `false`.

a) Fertigen Sie "vorher-nachher"-Beispielskizzen der Liste an, die Sie wie in den Folien K7 zur Vorlesung visualisieren – für die Methoden

   - `public void addFirst(int x)`
   - `public boolean dropFirst()`
   - `public void addLast(int x)`
   - `public boolean dropLast()`

   Dazu:

   - Zeichnen Sie jeweils geeignet (bzw. genügend) viele Listenelemente (die als Inhalt eingetragenen Zahlenwerte können Sie frei wählen) und alle relevanten Objektreferenzen (Verkettungen, `head`, benötigte Hilfsreferenzen wie z.B. ein Listencursor).
   - Unterscheiden Sie dabei jeweils drei "vorher"-Fälle: (i) leere Liste, (ii) einelementige Liste, (iii) Liste mit drei Elementen.
   - Bei den `add`-Methoden soll Ihre "vorher"-Zeichnung auch das neu hinzu zu fügende Element enthalten!

b) Implementieren Sie die Listenklasse:

- Ergänzen Sie in der mitgelieferten Datei `IntList.java` den fehlenden Code für die Methoden wie oben angegeben. Das Coding der vier in a) betrachteten Methoden sollte, wenn Sie die Diagramme vollständig beschriftet haben, direkt aus den Skizzen ableitbar sein.
- Fügen Sie der Klasse `IntList` eine Methode `public String toString()` hinzu. Die String-Repräsentation einer Liste soll immer mit einer öffnenden eckigen Klammer beginnen, danach die Elemente durch Komma und Leerzeichen getrennt enthalten und mit einer schließenden eckigen Klammer enden. Die Liste, welche die Elemente 1, 2 und 3 (in dieser Reihenfolge) enthält, sollte daher die Repräsentation `[1, 2, 3]` besitzen, die leere Liste die Repräsentation `[]`.
- Fügen Sie der Klasse `IntList` einen Iterator, wie in der Vorlesung beschrieben, hinzu, der es erlaubt vorwärts durch die Liste zu gehen.

  Hinweis: Hier soll nicht `java.util.Iterator` implementiert werden, sondern ein eigener Iterator nur für Ihre Listenklasse – Sie benötigen also keine `import`-Statements und keine Generics.
- **(optional)** Fügen Sie der Klasse `IntList` eine Methode `public void reverse()` hinzu. Diese soll eine Liste umkehren (d.h. das erste Element wird zum letzten etc.). Ihre Implementierung soll keinen neuen Speicherplatz mittels `new` anfordern, d.h. es soll sich um einen *in-place*-Algorithmus handeln.

c) Prüfen Sie die generelle Funktionsweise der Liste, indem Sie die mitgelieferte Klasse `IntListApp` ausführen. Falls Sie `reverse()` nicht programmiert haben, kommentieren Sie die entsprechende Stelle in der App-Klasse aus.

d) Testen Sie Ihre Liste mit der mitgelieferten Klasse `IntListTest` und beheben Sie eventuelle Fehler.

Aufgabe 2 (Sichtbarkeit von Variablen, Klassen und Methoden):

In Ihrer Implementierung für Aufgabe 1 sind verschiedene Elemente mit den Sichtbarkeitsmodifikatoren `public` oder `private` markiert. Begründen Sie, warum Sie diese so für sinnvoll erachten oder passen Sie die Modifikatoren gegebenenfalls an.