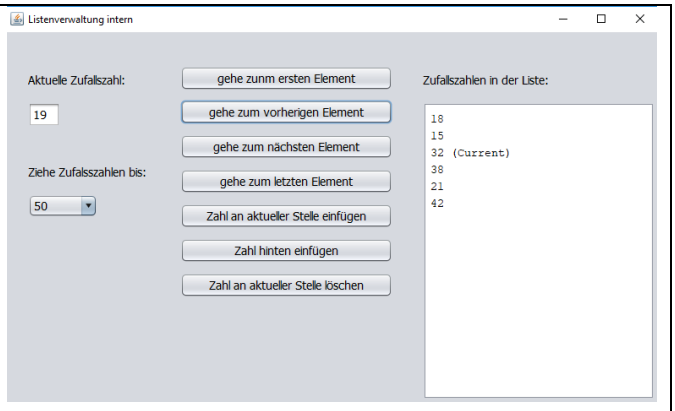
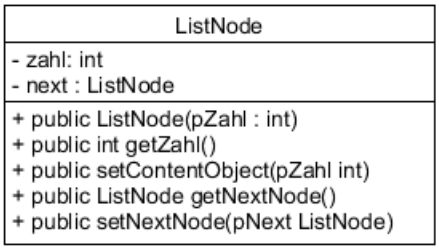
Projekt 14 Liste intern

In diesem Projekt geht es darum, die Funktionen einer möglichst einfachen Liste, nämlich eine Liste die Integerzahlen verwaltet, selbst zu programmieren. Insbesondere geht es dabei um die Methoden **append (zahl)**, **insert (zahl)**, **remove()**. Öffne zunächst das Projekt „Listenverwaltung Intern“ aus dem Schülerverzeichnis. Mit sieben Buttons kannst du die wichtigsten Funktionen einer Liste auslösen: Einfügen und löschen von Elementen, verschieben der aktuellen Position „Current“. Eingefügt werden Zufallszahlen, wobei man den Bereich wählen kann, aus dem die Zahlen gezogen werden.



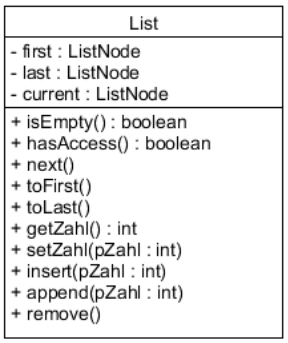
Ein Hinweis zur Benutzung des Programms: Fügt man eine erste Zahl in eine leere Liste ein, wird zunächst kein Current gesetzt. Das weitere Einfügen einer Zahl mit insert (zahl) wird erst möglich, wenn man z. B. den Current an die erste Position gesetzt hat.

Die Klasse ListNode enthält die notwendigen Methoden, die man zur Verwaltung eines Knotens benötigt: Jeder Knoten besitzt zwei Attribute: eine Intergerzahl **zahl** und den **Verweis** auf den Nachfolgeknoten next. Es gibt zwei Methoden zum Erfragen bzw. Setzen der Zahl : **getZahl()** und **setContentObject (pZahl int)**. Außerdem gibt es eine Methode, die den Verweis auf den Nachfolgeknoten liefert und mit der man diesen Verweis setzen kann: **getNextNode()** und **setNextNode (pNext ListNode)**.

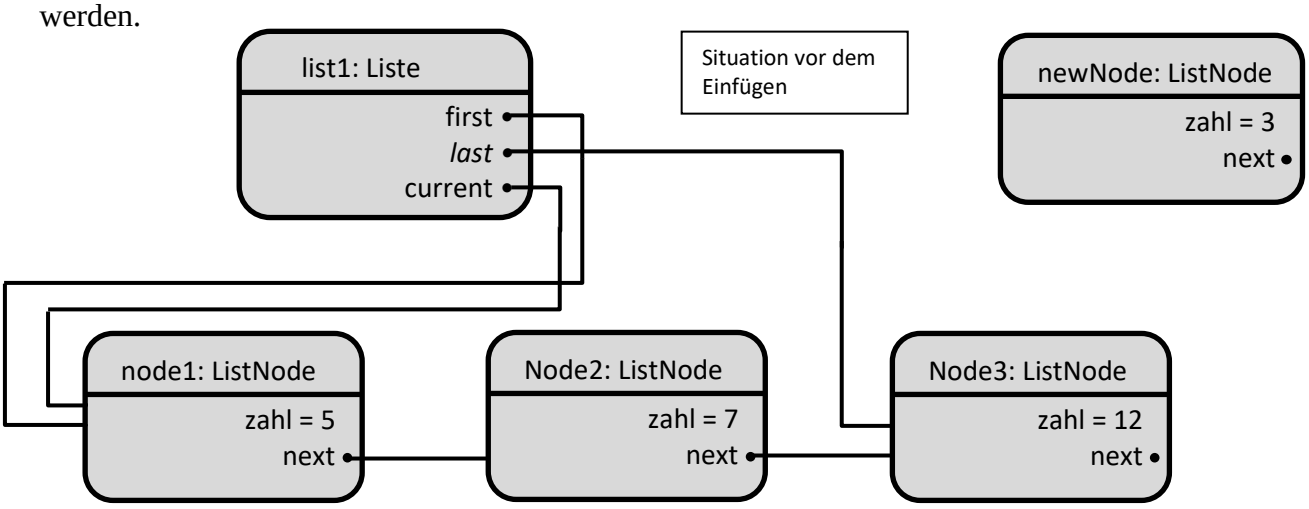


Die Klasse List baut auf der Klasse ListNode auf, indem die zu verwaltende Liste aus mehreren Knoten (ListNode) aufgebaut wird. Die bekannten Methoden isEmpty(), hasAccess(), next(), toFirst(), toLast(), getZahl(), setZahl (int pZahl) sind bereits fertig implementiert, du kannst dich am Quelltext orientieren um zu sehen, wie das Zusammenspiel zwischen den Klassen ListNode und List funktioniert.

Aufgabe 1: Deine erste Aufgabe ist es, die Methode **insert (pZahl : int)** zu implementieren, die einen neuen Knoten mit **pZahl** in die Liste **vor** der Position einfügen soll, die durch **current** angegeben wird. Die folgende Grafik zeigt, dass du dabei zwei Fälle unterscheiden musst:



Fall 1: **current** stimmt mit **first** überein, d. h. der neue Knoten soll am Beginn der Liste eingefügt werden.



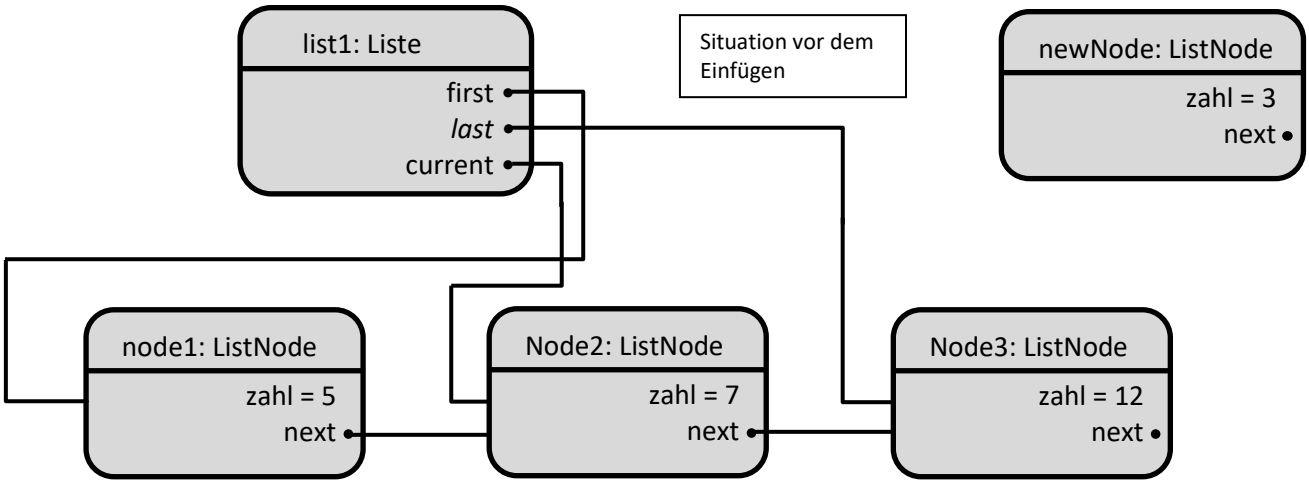
Folgende Operationen sind dazu notwendig:

1.) Lege einen neuen Knoten **newNode** mit dem **new**-Befeehl an, schreibe die neue Zahl hinein
2.) Lege den Verweis **next** von **newNode** auf **node1** (**newNode** wird der neue erste Knoten)
3.) Lege die Verweise **first** und **current** auf **newNode**

Zeichne die neuen Vereise als rote Linien in das Diagramm ein, implementiere anschließend für die Methode insert in der Klasse **Liste** den Fall 1.

Wenn du deine Implementation testen willst, musst du an eine Besonderheit denken: Wenn du das erste Element eingefügt hast, zeigt current noch nicht auf dieses Element. Die aktuelle Position musst du nach dem Einfügen auf das erste Element setzen (gehe zum ersten Element), erst danach kannst du ein weiteres Element einfügen.

Fall 2: **current** stimmt nicht mit **first** überein, d. h. der neue Knoten soll nicht am Beginn der Liste eingefügt werden. In der Abbildung ist dargestellt, dass der neue Knoten vor dem Knoten **node2** – also zwischen **node1** und **node2** eingefügt werden soll, **current** zeigt also auf **node2**.



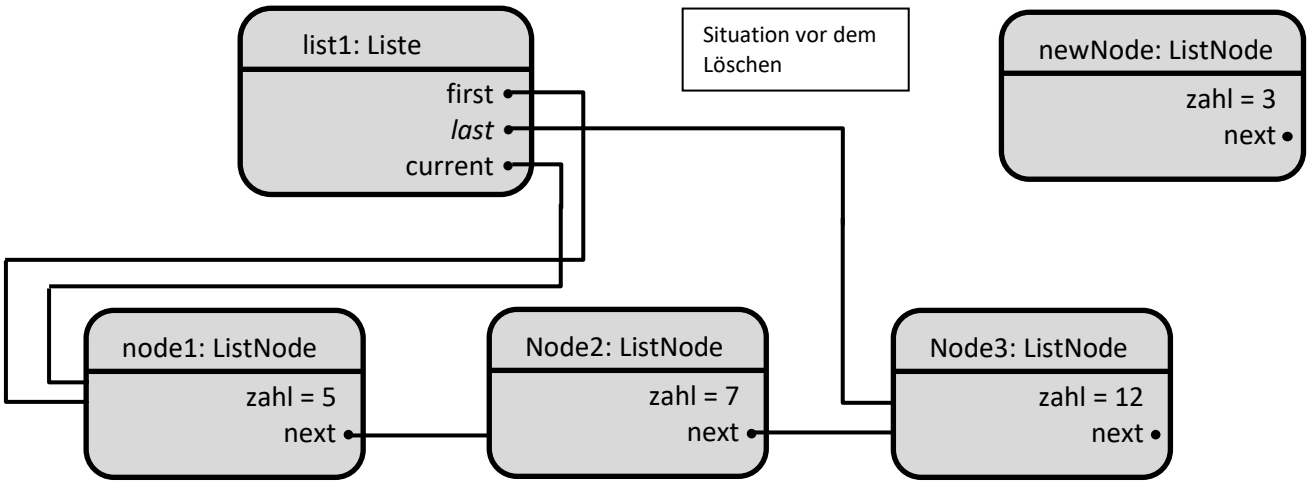
Die Klasse List enthält ganz am Ende die Methode **getPrevious(ListNode pNode)**. Diese Methode liefert zu einem Knoten **pNode** – falls dieser existiert - den Vorgängerknoten, also den Knoten, der vor dem Knoten **pNode** steht. Überlege, welche Schritte jetzt notwendig sind und wie du dabei die gerade vorgestellte Methode verwenden kannst. Zeichne wieder die neuen Vereise als rote Linien in das obige Diagramm ein, implementiere anschließend für die Methode insert in der Klasse **Liste** den Fall 2.

Fall 3: Es gibt noch einen Sonderfall, wenn die Liste, in die eingefügt werden soll, leer ist. Dieser Fall ist der einfachste, du musst nur einen neuen Knoten erzeugen und die Zeiger **current**, **first** und **last** richtig setzen. Implementiere auch diesen Fall.

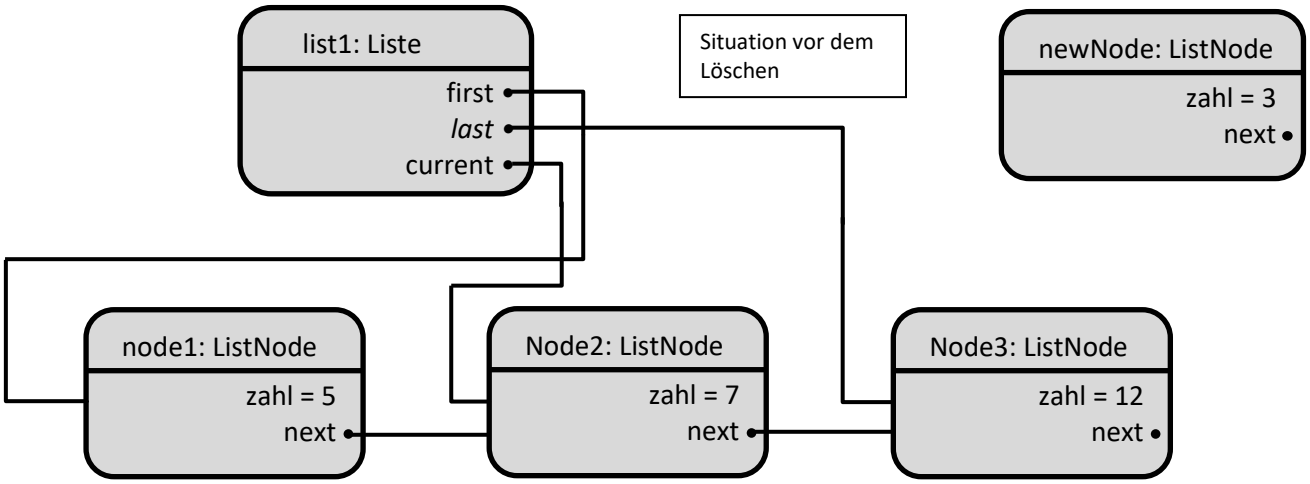
Aufgabe 2: Die Methode **append(pZahl)** hängt am Ende der Liste einen neuen Knoten an. Ist die Liste leer, kann die Methode **insert (pZahl)** benutzt werden, ansonsten kann das Anhängen ohne weitere Fallunterscheidung implementiert werden. Überlege, wie die Zeiger verändert werden müssen und implementiere.

Aufgabe 3: Die Methode **remove ()** löscht bekanntlich den Knoten, auf den **current** aktuell zeigt. Auch hier müssen wieder zwei Fälle unterschieden werden:

Fall 1: **current** stimmt mit **first** überein, d. h. der erste Knoten der Liste soll gelöscht werden.



Fall 2: **current** stimmt nicht mit **first** überein, d. h. der Knoten, der gelöscht werden soll, steht nicht am Beginn der Liste. In der folgenden Abbildung ist wieder dargestellt, dass der zweite Knoten **node2** gelöscht werden soll, **current** zeigt also auf **node2**.



Teste zum Schluss deine gesamte Implementation aus.

Aufgabe 4: Vergleiche deine Implementation einer Liste mit der Liste aus den Abiturimplementationen. Welche Unterschiede gibt es? Nenne mindestens zwei.