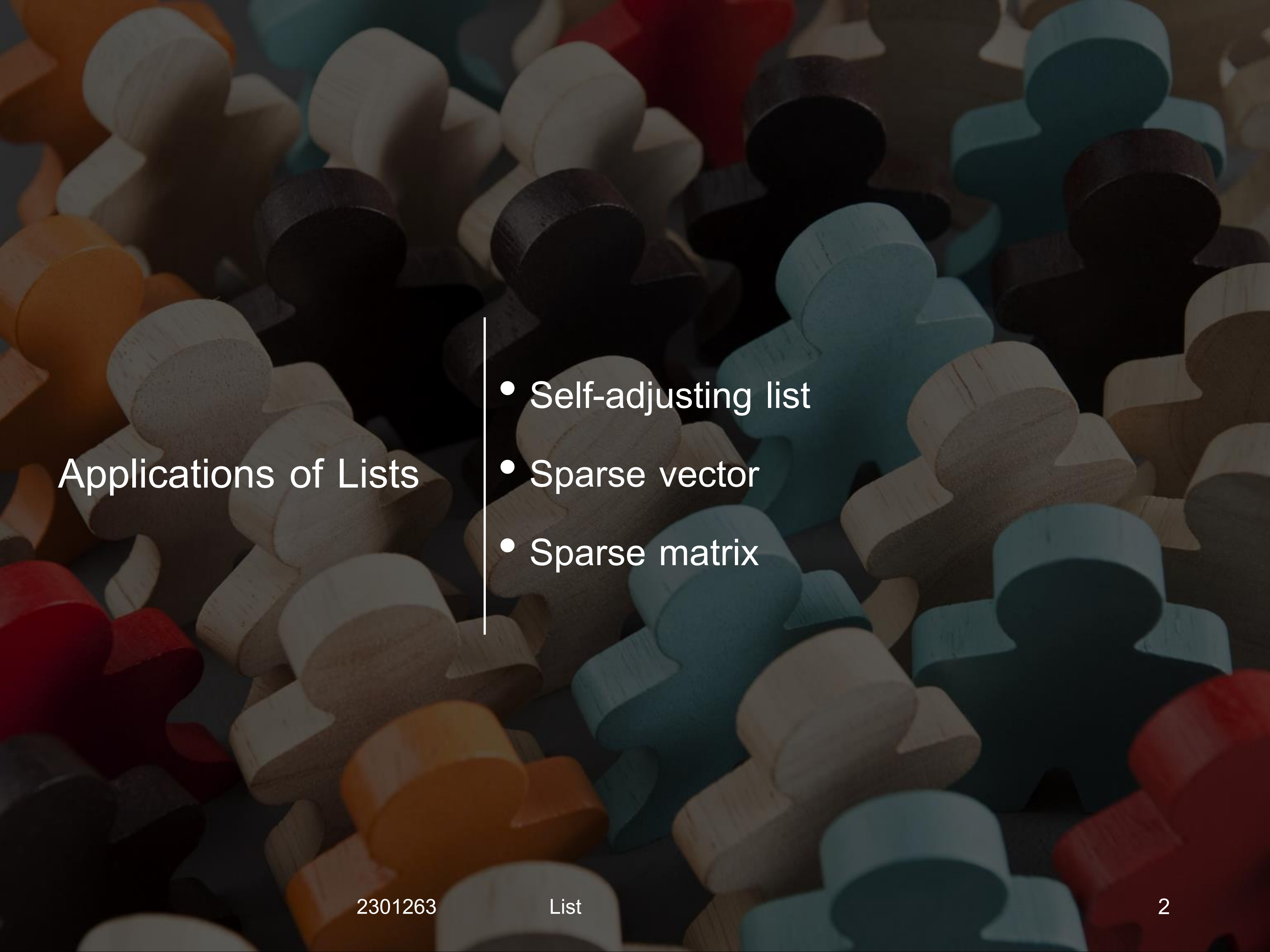
List (รายการ)

Applications

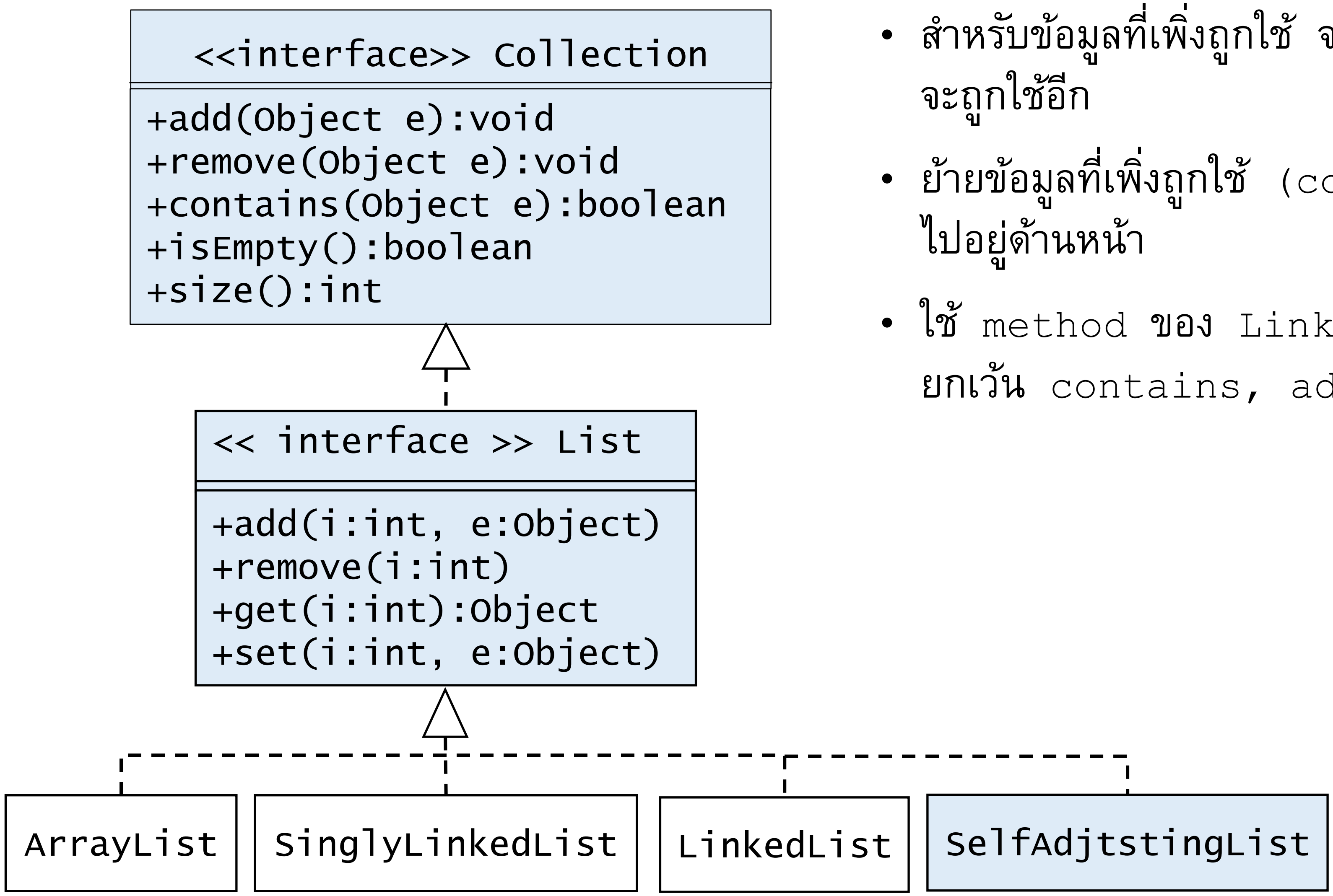


Applications of Lists

- Self-adjusting list
- Sparse vector
- Sparse matrix

Self-adjusting Lists

Self-adjusting Lists



- สำหรับข้อมูลที่เพิ่งถูกใช้ จะมีโอกาสสูงที่จะถูกใช้อีก
- ย้ายข้อมูลที่เพิ่งถูกใช้ (contains) ไปอยู่ด้านหน้า
- ใช้ method ของ LinkedList ได้ ยกเว้น contains, add

Class SelfAdjustingList

```
public class SelfAdjustingList implements List {
  private static class LinkedNode { … }
  private LinkedNode header;
  private int size;
  public SelfAdjustingList() { … }
  public boolean contains(Object e) { … }
  private LinkedNode nodeOf(Object e) { … }
  private void addBefore(LinkedNode q,Object e){ … }
  public void add(Object e) { … }
}
```

Class LinkedNode

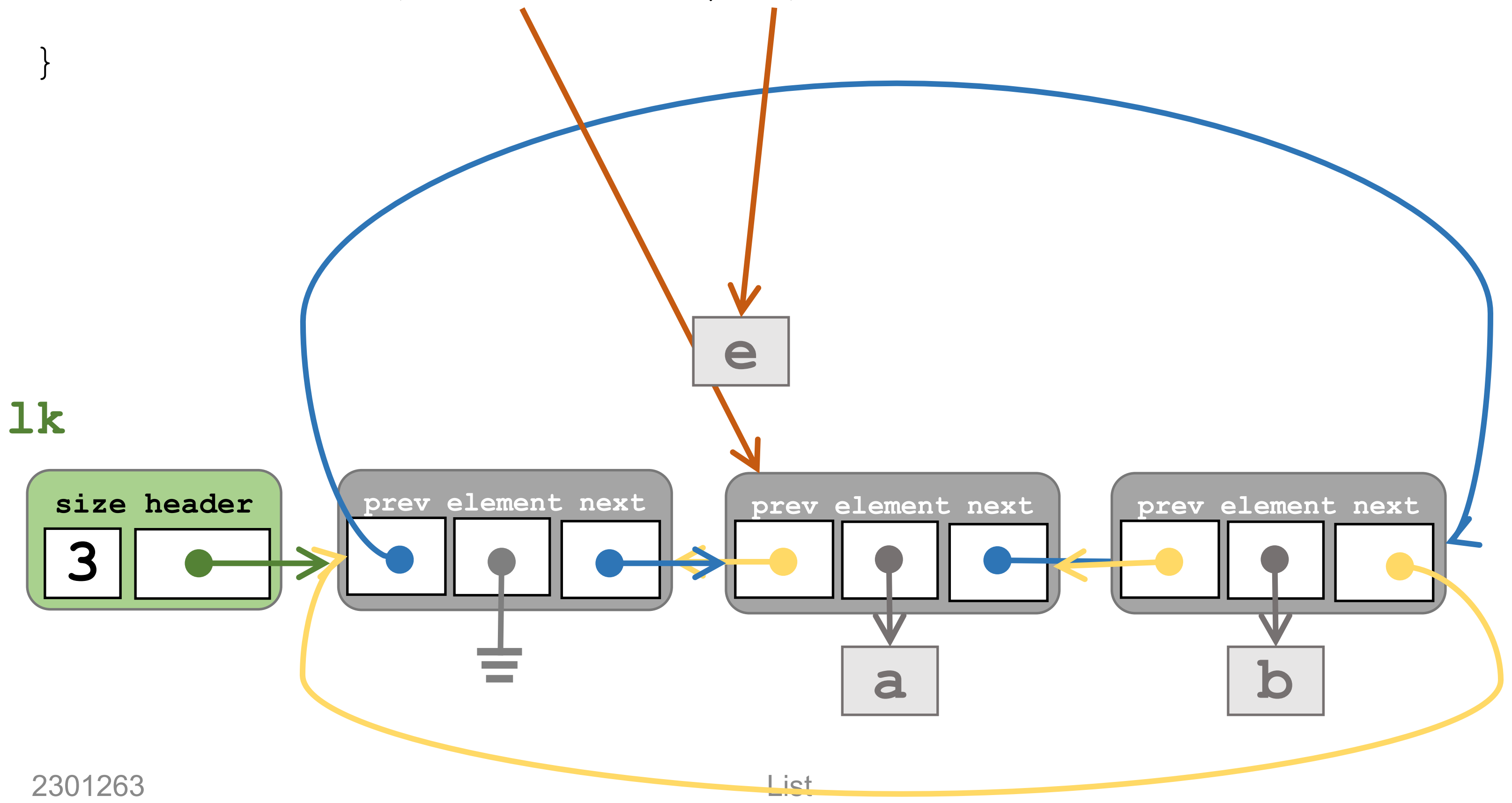
```
public class SelfAdjustingList implements List {
 private static class LinkedNode {
    Object element;
    LinkedNode prev, next;
    LinkedNode(Object e, LinkedNode p, LinkedNode n) {
      this.element = e;
      this.prev = p;
      this.next = n;
    }
  }
…}
```

Method add

Class SelfAdjustingList

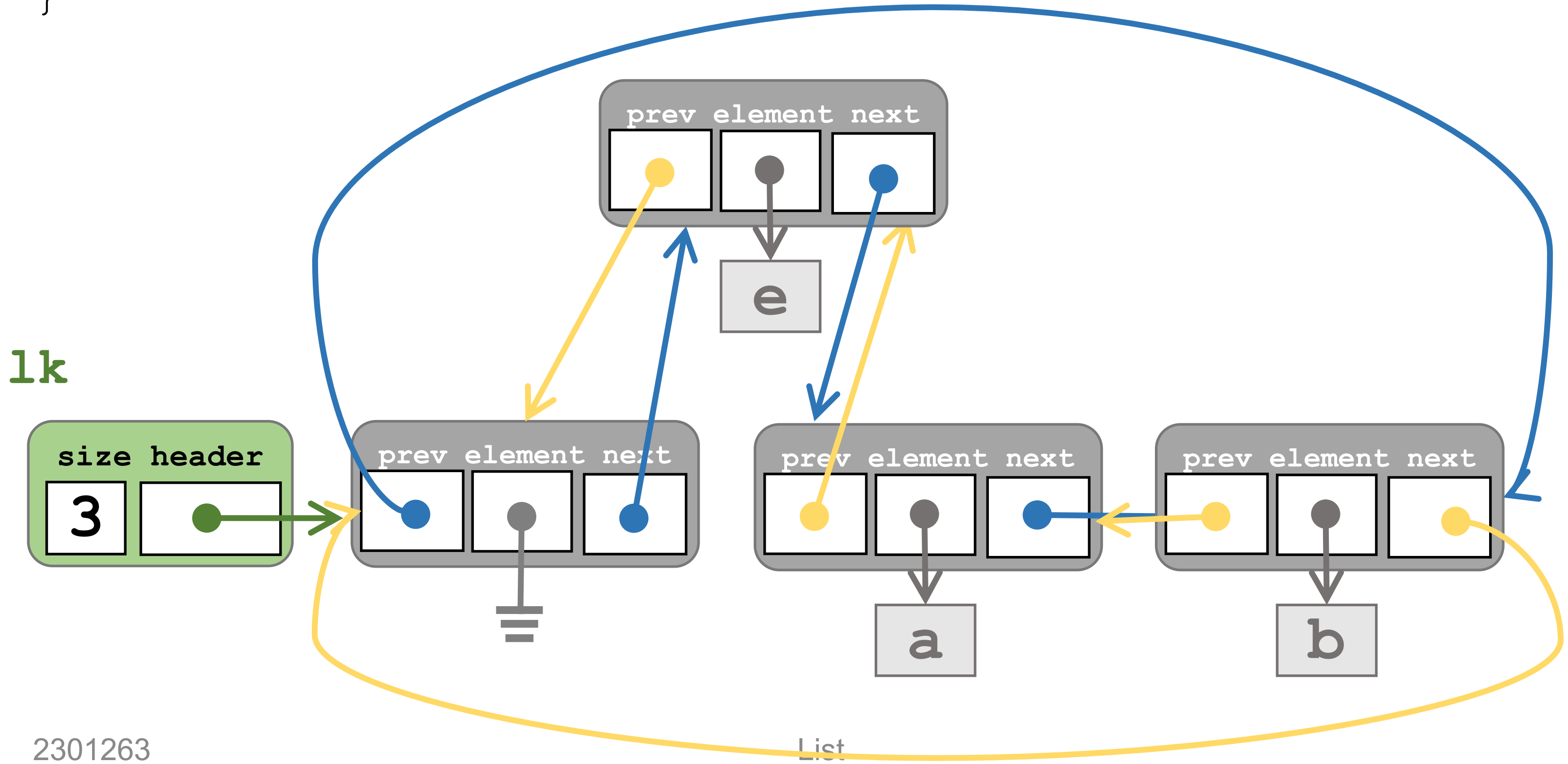
Add: Class SelfAdjustingList

```
public void add(Object e) {
    addBefore(header.next, e);
}
```



Add: Class SelfAdjustingList

```
public void add(Object e) {
    addBefore(header.next, e);
}
```

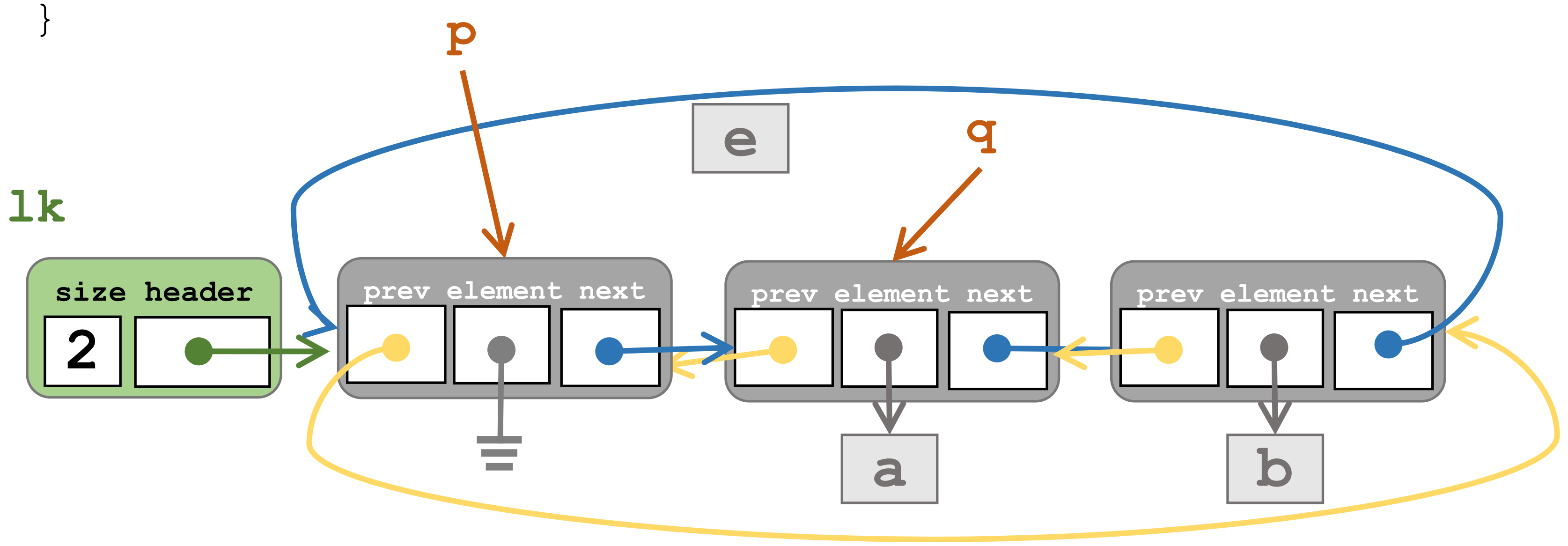


Method addBefore

Class SelfAdjustingList

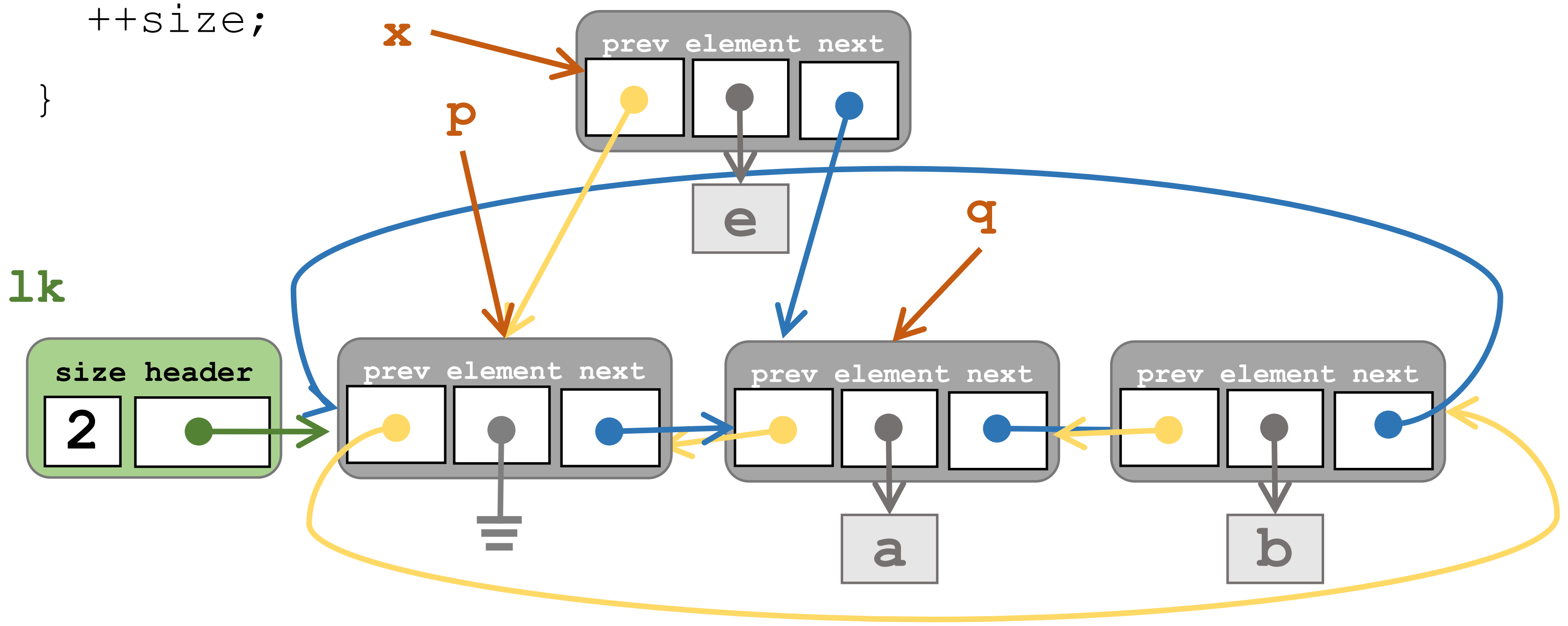
AddBefore: Class SelfAdjustingList

```
private void addBefore(LinkedNode q, Object e) {
  LinkedNode p = q.prev;
  LinkedNode x = new LinkedNode(e, p, q);
  p.next = q.prev = x;
  ++size;
}
```



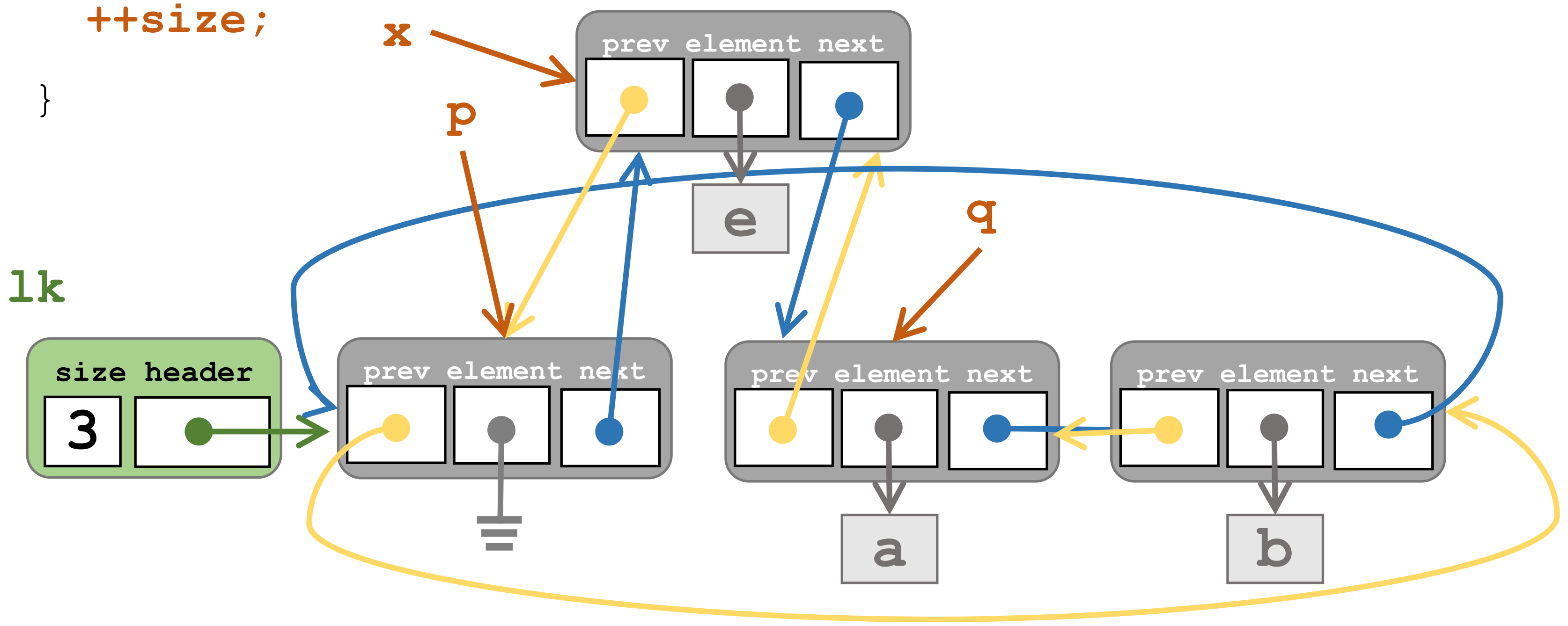
AddBefore: Class SelfAdjustingList

```
private void addBefore(LinkedNode q, Object e) {
  LinkedNode p = q.prev;
  LinkedNode x = new LinkedNode(e, p, q);
  p.next = q.prev = x;
  ++size;
}
```



AddBefore: Class SelfAdjustingList

```
private void addBefore(LinkedNode q, Object e) {
  LinkedNode p = q.prev;
  LinkedNode x = new LinkedNode(e, p, q);
  p.next = q.prev = x;
  ++size;
}
```

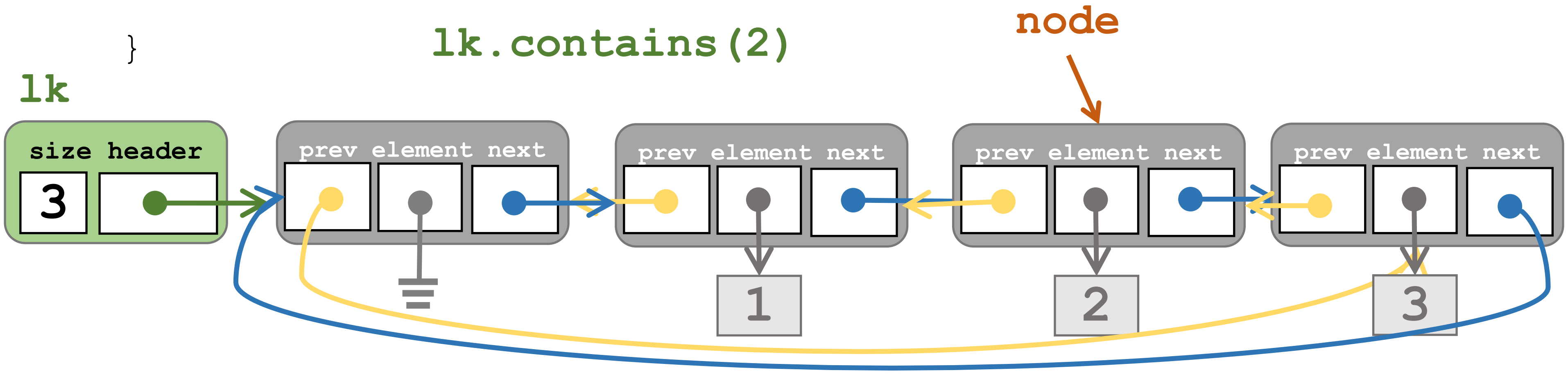


Method contains

Class SelfAdjustingList

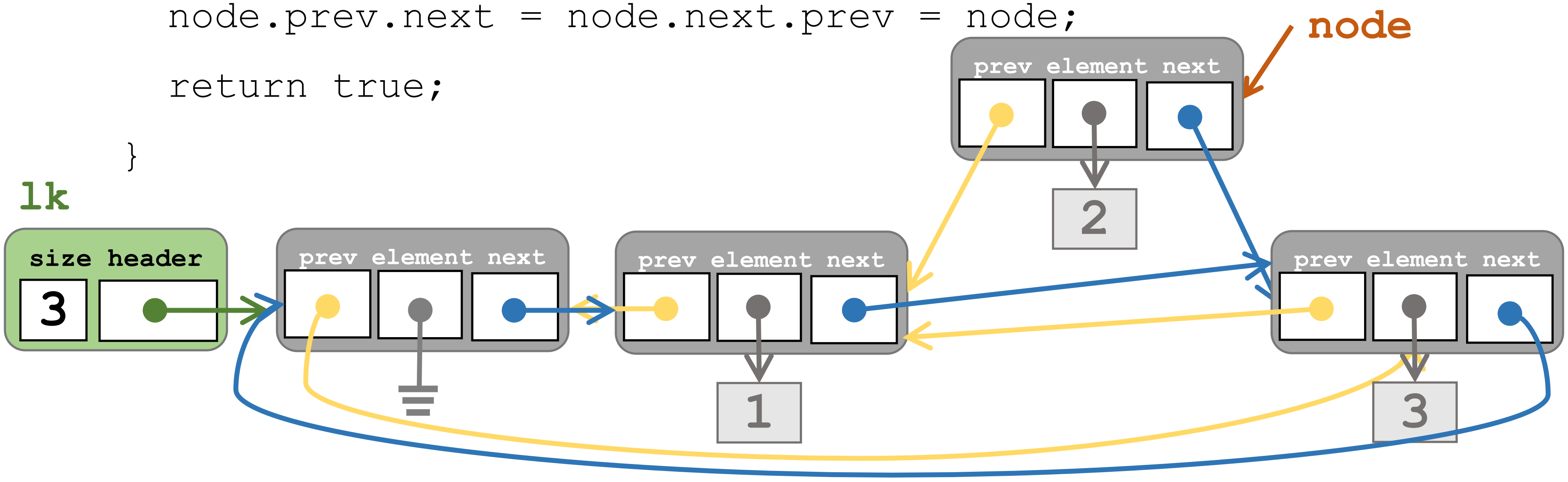
Contains: Class SelfAdjustingList

```
public boolean contains(Object e) {
  LinkedNode node = nodeOf(e);
  if (node==header) return false;
  node.prev.next=node.next;   node.next.prev=node.prev;
  node.prev = header;         node.next = header.next;
  node.prev.next = node.next.prev = node;
  return true;
}
```



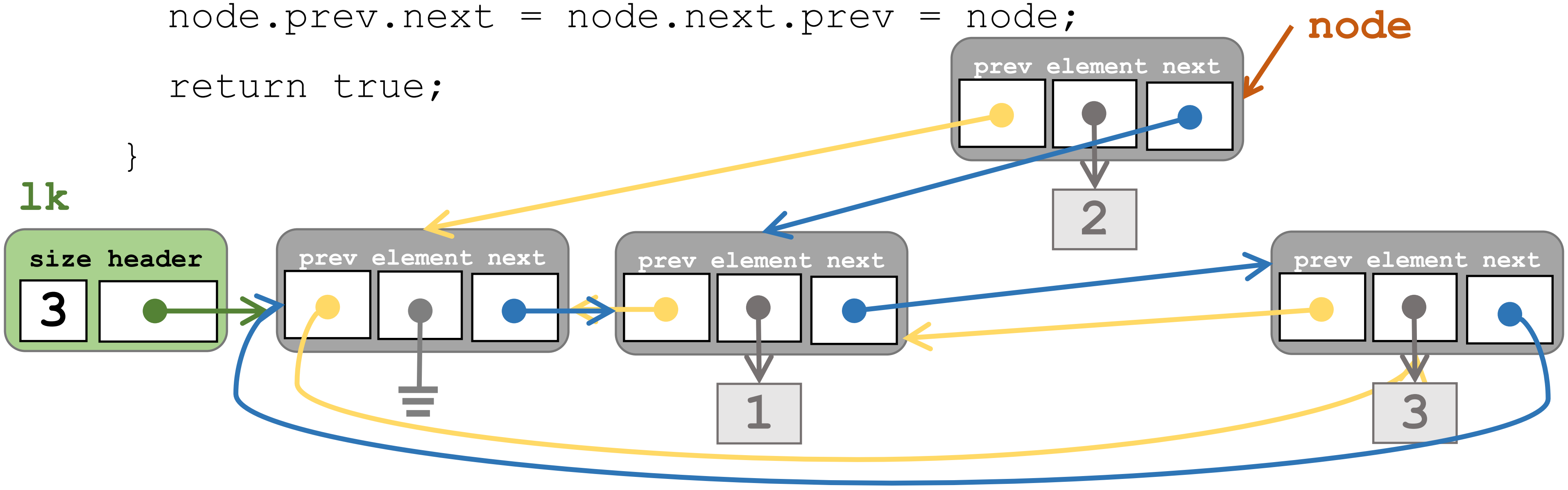
Contains: Class SelfAdjustingList

```
public boolean contains(Object e) {
  LinkedNode node = nodeOf(e);
  if (node==header) return false;
  node.prev.next=node.next;    node.next.prev=node.prev;
  node.prev = header;          node.next = header.next;
  node.prev.next = node.next.prev = node;
  return true;
}
```



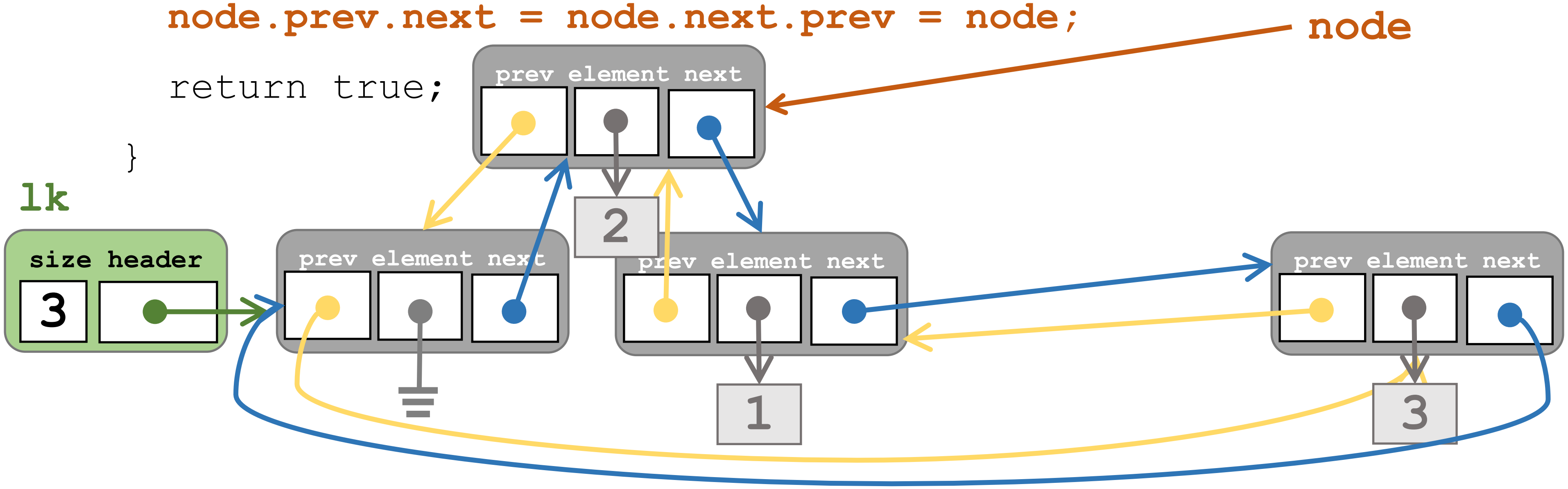
Contains: Class SelfAdjustingList

```
public boolean contains(Object e) {
  LinkedNode node = nodeOf(e);
  if (node==header) return false;
  node.prev.next=node.next;   node.next.prev=node.prev;
  node.prev = header;         node.next = header.next;
  node.prev.next = node.next.prev = node;
  return true;
}
```



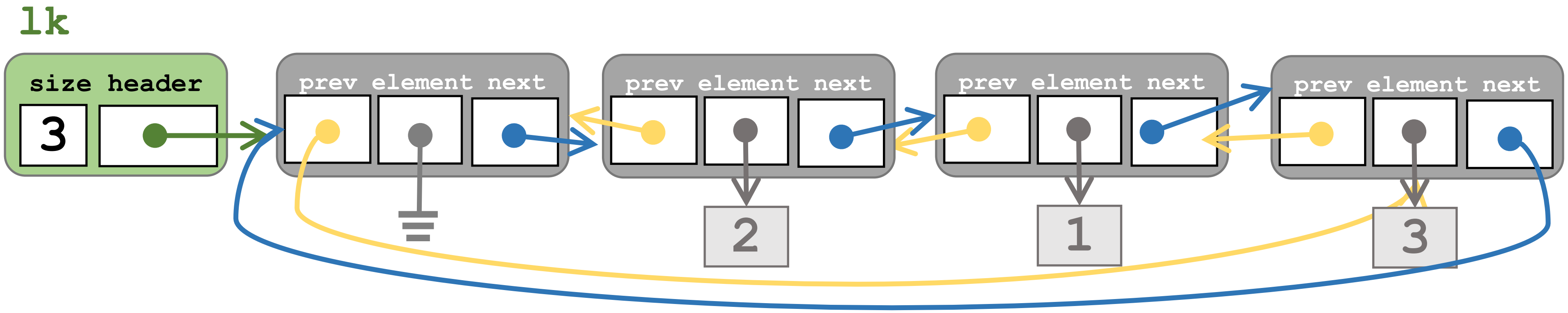
Contains: Class SelfAdjustingList

```
public boolean contains(Object e) {
  LinkedNode node = nodeOf(e);
  if (node==header) return false;
  node.prev.next=node.next;   node.next.prev=node.prev;
  node.prev = header;         node.next = header.next;
  node.prev.next = node.next.prev = node;
  return true;
}
```



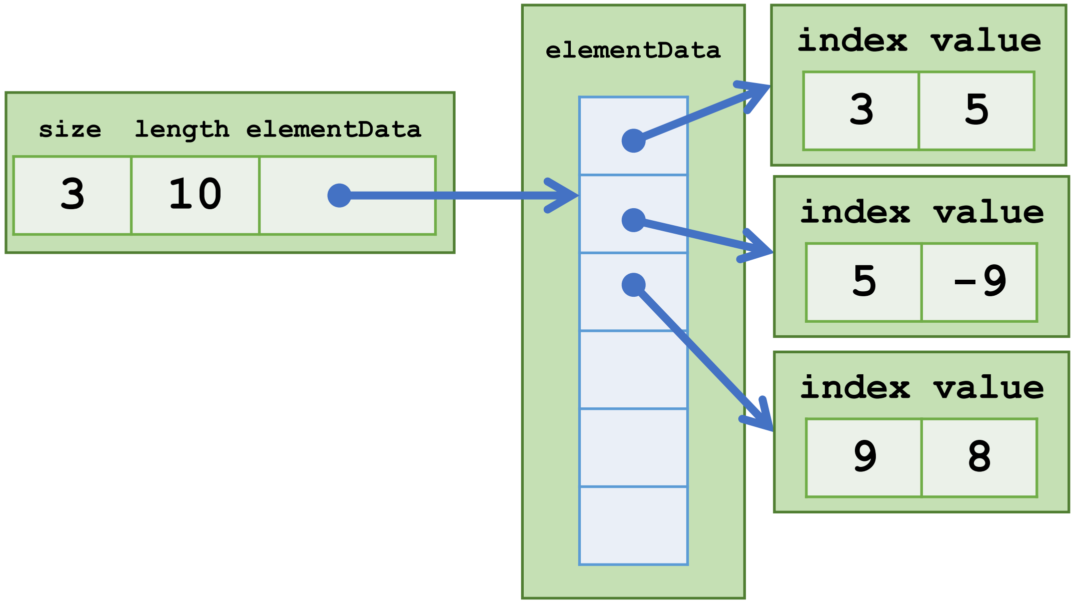
Contains: Class SelfAdjustingList

```
public boolean contains(Object e) {
  LinkedNode node = nodeOf(e);
  if (node==header) return false;
  node.prev.next=node.next;    node.next.prev=node.prev;
  node.prev = header;          node.next = header.next;
  node.prev.next = node.next.prev = node;
  return true;
}
```

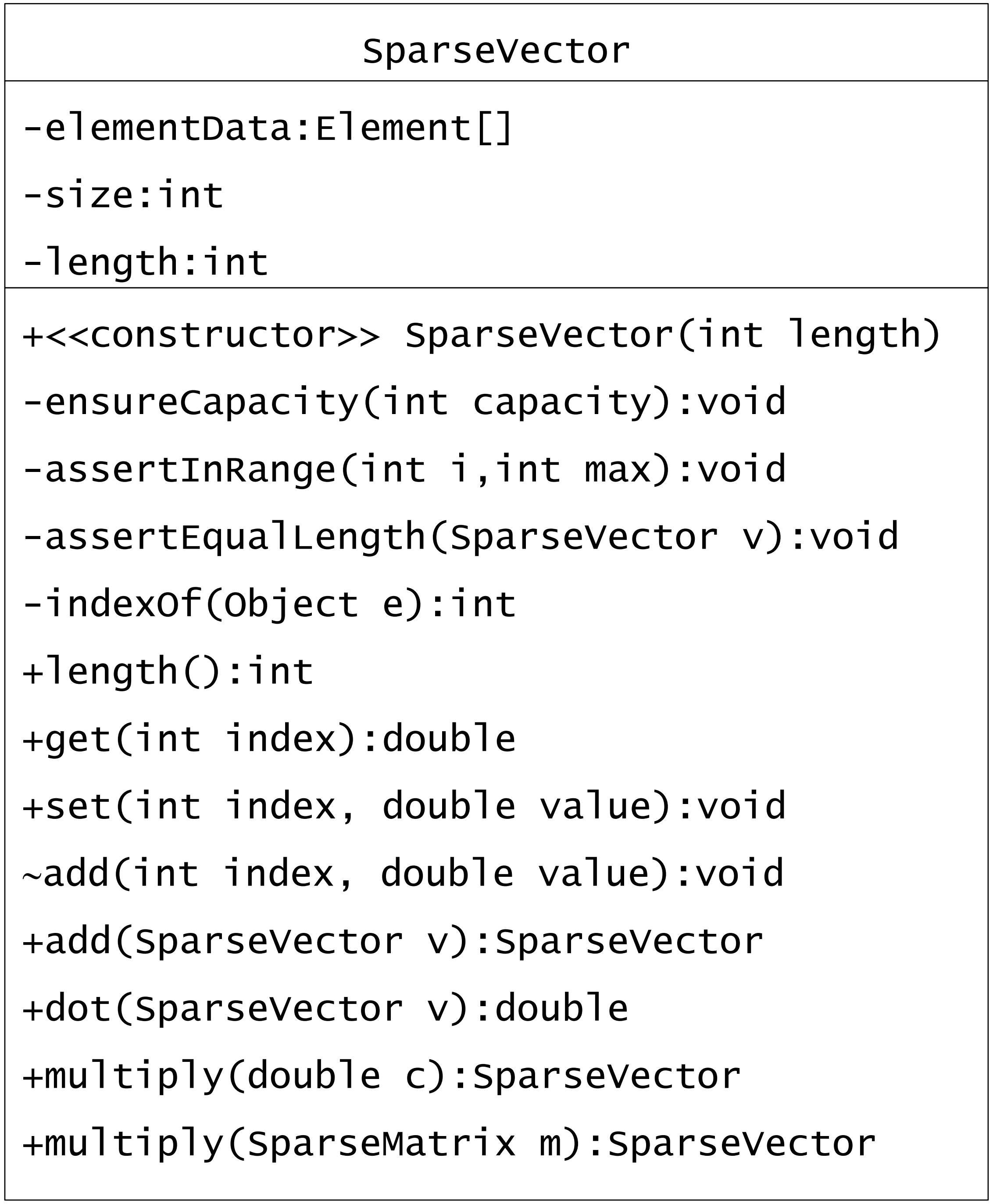


Sparse Vector

Sparse Vector



[0, 0, 0, 5, 0, -9, 0, 0, 0, 8]



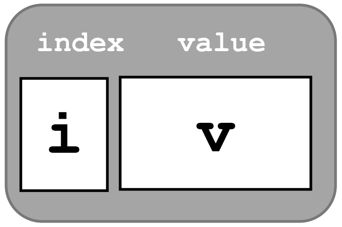
```
SparseVector
-elementData:Element[]
-size:int
-length:int
+<<constructor>> SparseVector(int length)
-ensureCapacity(int capacity):void
-assertInRange(int i,int max):void
-assertEqualLength(SparseVector v):void
-indexOf(Object e):int
+length():int
+get(int index):double
+set(int index, double value):void
~add(int index, double value):void
+add(SparseVector v):SparseVector
+dot(SparseVector v):double
+multiply(double c):SparseVector
+multiply(SparseMatrix m):SparseVector
```

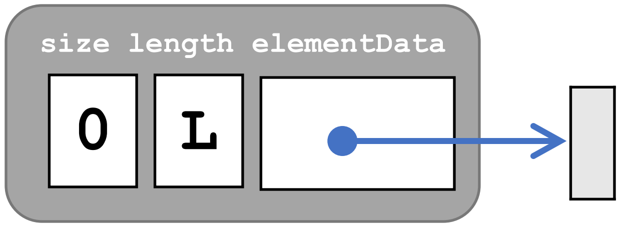
Methods in Class SparseVector

```
public class SparseVector {
   private static class Element { ... }
   private Element[] elementData;
   private int size;
   private int length;
   public SparseVector() {...}
   public int length() {...}
   public double get(int index) {...}
   private void assertInRange(int index) {...}
   public void set(int index, double value) {...}
   private void assertEqualLength(SparseVector v) {...}
   void add(int i, int index, double value) {...}
   public SparseVector add(SparseVector v) {...}
   void append(int index, double value) {...}
   public SparseVector multiply(double c) {...}
   public SparseVector multiply(SparseMatrix m) {...}
   public SparseVector dot(SparseVector v) {...}
}
```

Create new object: Class SparseVector

```
public class SparseVector {
  private static class Element {
    int index;
    double value;
    Element(int i, double v) {
      this.index = i;  this.value = v;
    }
  }
  private int size;
  private int length;
  private Element[] elementData;
  public SparseVector(int length) {
    this.elementData = new Element[0];
    this.size = 0;
    this.length = length;
  }
… }
```





Method get

Class SparseVector

get: Class SparseVector

```
public double get(int index) {
  assertInRange(index);
  for(int i=0; i<size; i++) {
    if (elementData[i].index == index)
      return elementData[i].value;
    if (elementData[i].index > index) break;
  }
  return 0.0;
}
private void assertInRange(int index) {
  if (index<0 || index>=length)
    throw new IndexOutOfBoundsException()
}
```

Method set

Class SparseVector

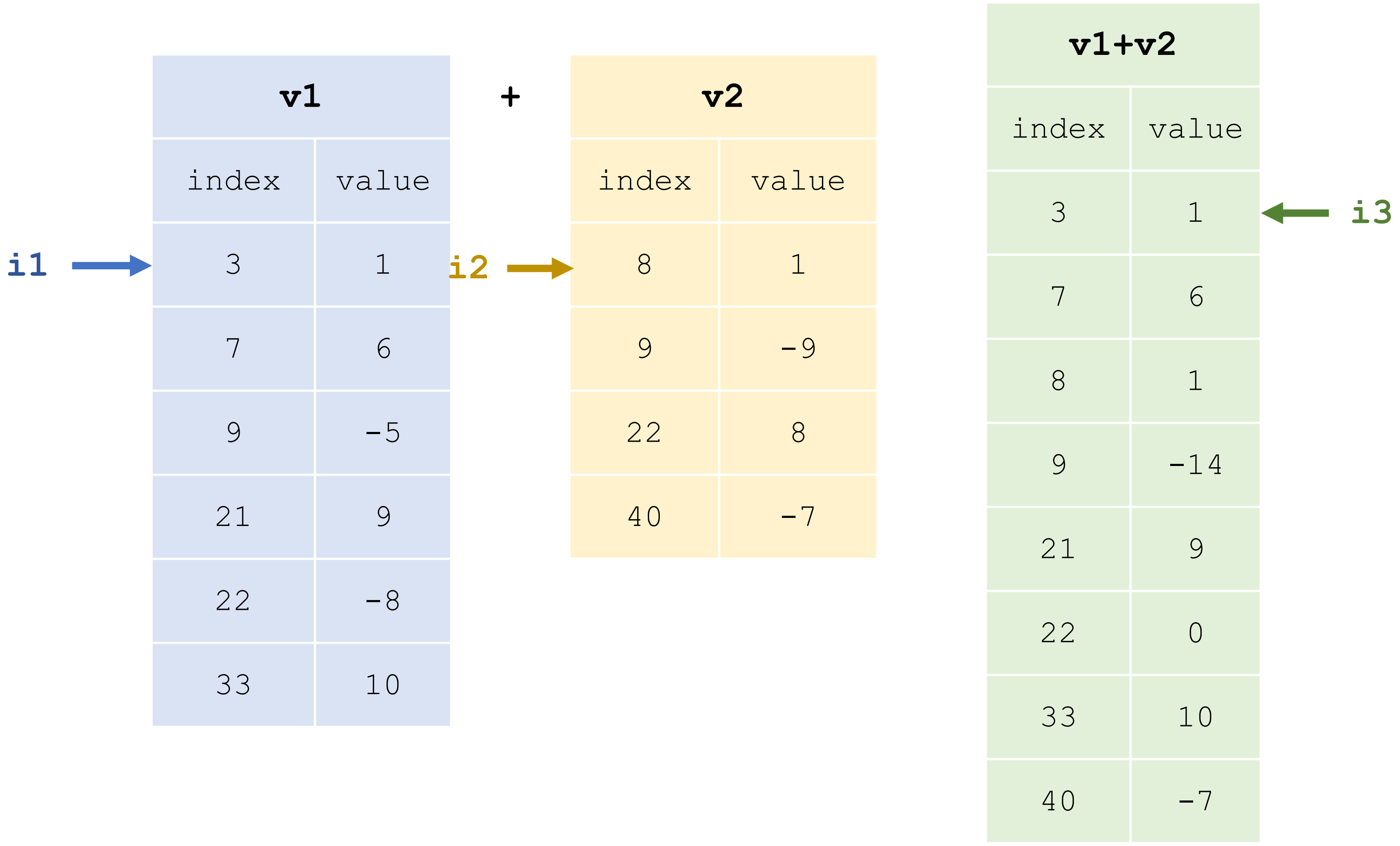
set: Class SparseVector

```
public void set(int index, double value) {
  int i = 0;
  while (i<size && elementData[i].index<index)  i++;
  if (i<size && elementData[i].index == index)
    elementData[i].value = value;
  else
    add(i, index, value);
}
void add(int i, int index, double value) {
  if (value != 0) {
    ensureCapacity(size+1);
    for (int k=size; k>i; k--)  elementData[k] = elementData[k-1];
    elementData[i] = new Element(index, value);
    ++size;
  }
}
```

Method add

Class SparseVector

Adding Sparse Vectors



v1	
index	value
3	1
7	6
9	-5
21	9
22	-8
33	10

+

v2	
index	value
8	1
9	-9
22	8
40	-7

v1+v2	
index	value
3	1
7	6
8	1
9	-14
21	9
22	0
33	10
40	-7

add: Class SparseVector

```
public SparseVector add(SparseVector v2) {
  SparseVector v1 = this;
  SparseVector v3 = new SparseVector(v1.length());
  int i1 = 0, i2 = 0, i3 = 0;
  while (i1 < v1.size && i2 < v2.size) {
    Element e1 = v1.elementData[i1];    Element e2 = v2.elementData[i2];
      if (e1.index < e2.index) {
        v3.add(i3++, e1.index, e1.value);          i1++;          }
      else if (e1.index > e2.index) {
        v3.add(i3++,e2.index, e2.value);           i2++;          }
      else {
        v3.add(i3++,e1.index, e1.value+e2.value);  i1++;  i2++;  }
  }
  while (i1 < v1.size) {
    Element e1 = elementData[i1++];   v3.add(i3++,e1.index, e1.value); }
  while (i2 < v2.size) {
    Element e2 = elementData[i2++];   v3.add(i3++,e2.index, e2.value); }
  return v3;
}
```

Method dot

Class SparseVector

Dot Sparse Vectors

v1.elementData	
index	value
3	1
7	6
9	-5
21	9
22	-8
33	10

.

v2.elementData	
index	value
8	1
9	-9
22	8
40	-7

v1	**v2**	**r**
3	8	0
7	8	0
9	8	0
9	9	45
21	9	45
21	22	45
22	22	-19
33	22	-19
33	40	-19

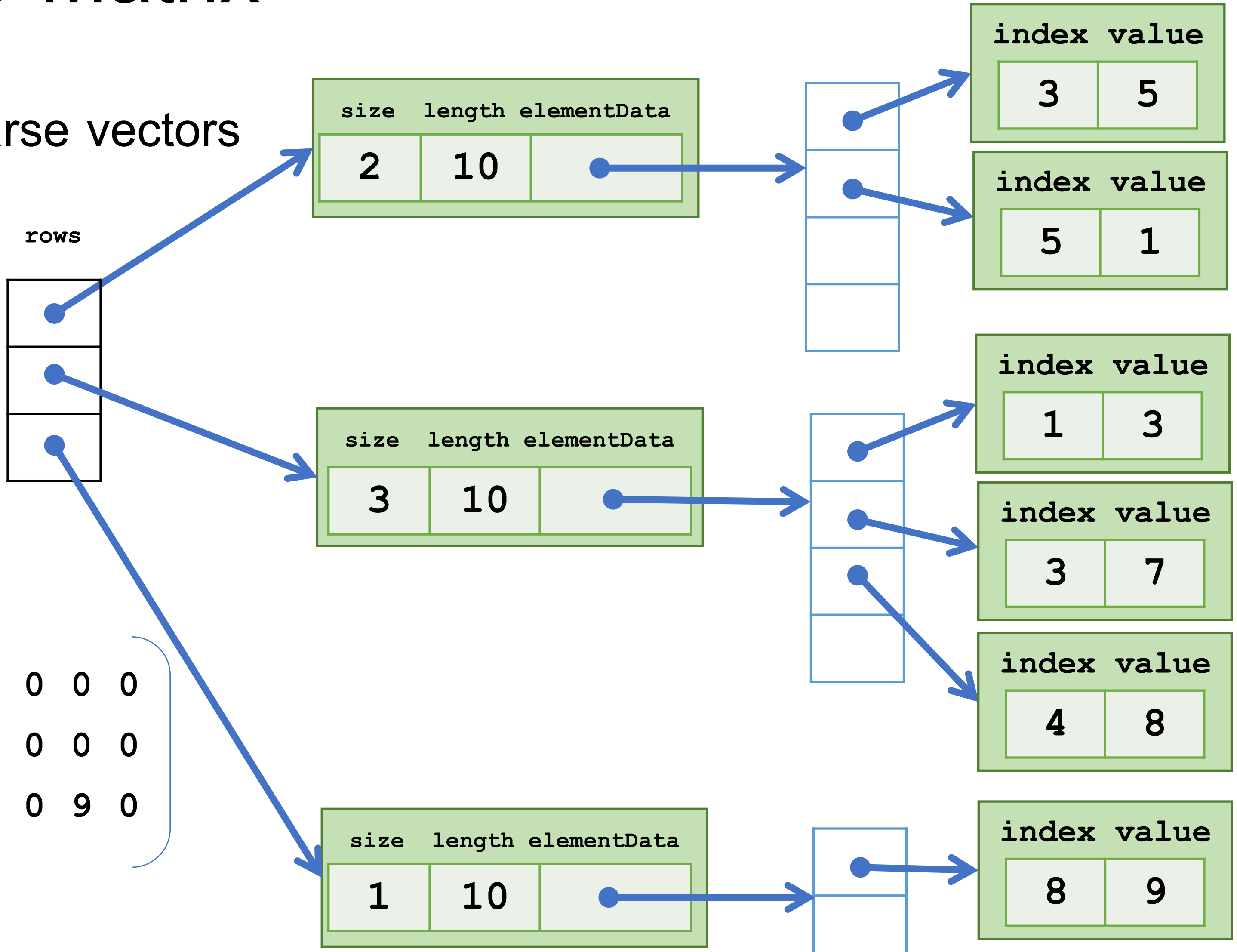
dot: Class SparseVector

```
public double dot(SparseVector v2) {
  assertEqualLength(v2);
  SparseVector v1 = this;
  double r = 0;
  int i1 = 0, i2 = 0;
  while (i1 < v1.size && i2 < v2.size) {
    Element e1 = v1.elementData[i1];
    Element e2 = v2.elementData[i2];
    if      (e1.index < e2.index) i1++;
    else if (e1.index > e2.index) i2++;
    else {
      r += e1.value * e2.value;
      i1++; i2++;
    }
  }
  return r;
}
```

SparseMatrix

Sparse Matrix

array ของ sparse vectors



$$\begin{pmatrix} 0 & 0 & 0 & 5 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 7 & 8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 0 \end{pmatrix}$$

Class SparseMatrix

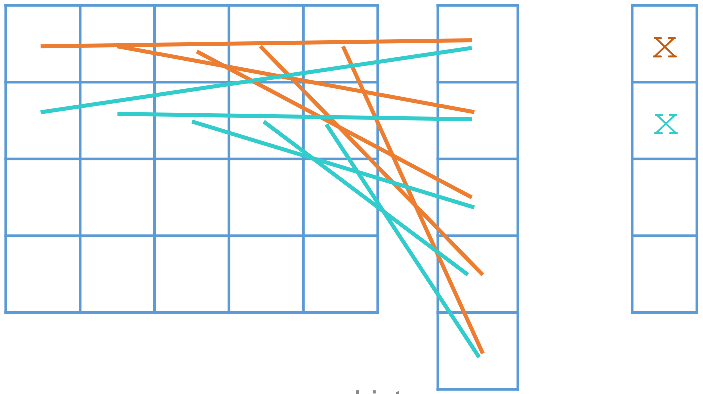
```
public class SparseMatrix {
  SparseVector[] rows;
  public SparseMatrix(int r, int c) {
    rows = new sparseVector[r];
    for (int i=0; i<r; i++)
      rows[i] = new SparseVector(c);
  }
  public int numRows() { return rows.length;      }
  public int numCols() { return rows[0].length(); }
  public void set(int r, int c, double v) {
    assertInRange(r,c); rows[r].set(c,v);  }
  public double get(int r, int c) {
    assertInRange(r,c); return rows[r].get(c);   }
  ...
}
```

add: Class SparseMatrix

```
public SparseMatrix add(SparseMatrix m2) {
  SparseMatrix m1 = this;
  int r = m1.numRows();
  int c = m1.numCols();
  if (r!=m2.numRows() || r!=m2.numCols())
    throw new IllegalArgumentException();
  SparseMatrix m3 = new SparseMatrix(r,c);
  for (int i=0; i<r; i++)
    m3.rows[i] = m1.rows[i].add(m2.rows[i]);
  return m3;
}
```

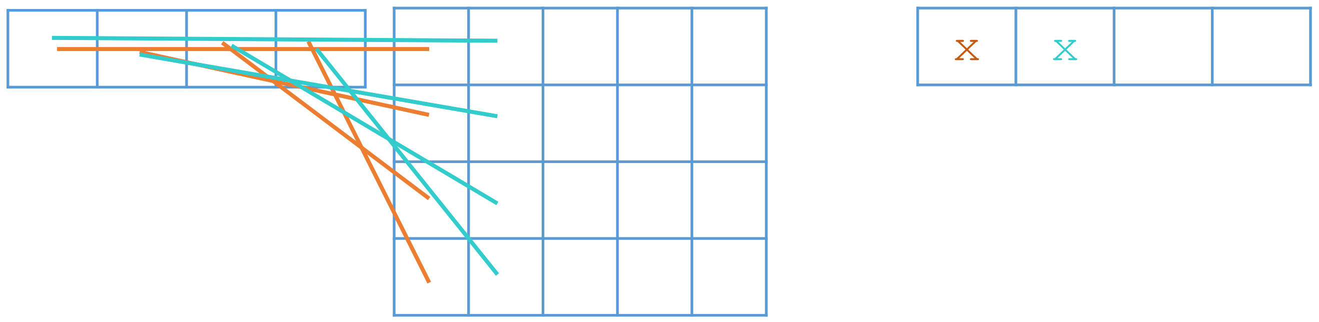
multiply: Class SparseMatrix

```
public SparseVector multiply(SparseVector v) {
  if (v.length() != this.numCols())
    throw new IllegalArgumentException();
  SparseVector r = new SparseVector(this.numRows());
  for (int i=0; i<this.numRows(); i++)
    r.set(i, rows[i].dot(v))
  return r;
}
```



multiply: Class SparseVector

```
public SparseVector multiply(SparseMatrix m) {
   if (this.length != m.numRows())
      throw new IllegalArgumentException();
   SparseVector r = new SparseVector(m.numCols());
   for(int i=0; i<this.length(); i++)
      r = r.add(m.rows[i].multiply(this.get(i)));
   return r;
}
```



multiply: Class SparseMatrix

```
public SparseMatrix multiply(SparseMatrix m2) {
  SparseMatrix m1 = this;
  if (m1.numCols() != m2.numRows())
    throw new IllegalAugumentException();
  SparseMatrix m3=new SparseMatrix(m1.numRows(),m2.numCols());
  for (int i=0; i<m1.numRows(); i++)
    m3.rows[i] = m1.rows[i].multiply(m2);
  return m3;
}
```