DM503
Programming B

Peter Schneider-Kamp

petersk@imada.sdu.dk

http://imada.sdu.dk/~petersk/DM503/

ABSTRACT DATATYPE FOR LISTS

List ADT: Specification

- data are all integers, here represented as primitive int
- operations are defined by the following interface

```
public interface ListOfInt {
    public int get(int i);                   // get i-th integer (0-based)
    public void set(int i, int elem);        // set i-th element
    public int size();                       // return length of list
    public void add(int elem);               // add element at end
    public void add(int i, int elem);        // insert element at pos. i
    public void remove(int i);               // remove i-th element
}
```

Partially Full Arrays

- arrays are fixed-length
- lists are variable-length
- **Idea:**
  - use an array of (fixed) length
  - track number of elements in variable
- **Example:** add(23) add(42) add(-3) remove(0) add(1, 23)

num 3

data 42 | 23 | -3 | |

List ADT: Design & Implementation 1

- Design 1: partially full arrays of int
- Implementation 1:

```
public class PartialArrayListOfInt implements ListOfInt {
    private int limit;              // maximal number of elements
    private int[] data;             // elements of the list
    private int num = 0;            // current number of elements
    public PartialArrayListOfInt(int limit) {
        this.limit = limit;
        this.data = new int[limit];
    }
    …
}
```

List ADT: Implementation 1

- Implementation 1 (continued):

```
public class PartialArrayListOfInt implements ListOfInt {   …
    private int[] data;
    private int num = 0;   …
    public int get(int i) {
        if (i < 0 || i >= num) {
            throw new IndexOutOfBoundsException();
        }
        return this.data[i];
    }
    …
}
```

List ADT: Implementation 1

- Implementation 1 (continued):

```
public class PartialArrayListOfInt implements ListOfInt {   …
    private int[] data;
    private int num = 0;   …
    public void set(int i, int elem) {
        if (i < 0 || i >= num) {
            throw new IndexOutOfBoundsException();
        }
        this.data[i] = elem;
    }
    …
}
```

List ADT: Implementation 1

- Implementation 1 (continued):

```
public class PartialArrayListOfInt implements ListOfInt {   …
    private int[] data;
    private int num = 0;   …
    public int size() {
        return this.num;
    }
    public void add(int elem) {
        this.add(this.num, elem);           // insert at end
    }
    …
}
```


UNIVERSITY OF SOUTHERN DENMARK.DK

List ADT: Implementation 1

- Implementation 1 (continued):

```
public class PartialArrayListOfInt implements ListOfInt {   …
    public void add(int i, int elem) {
        if (i < 0 || i > num) {  throw new Index…Exception();  }
        if (num >= limit) {  throw new RuntimeException("full!");  }
        for (int j = num-1; j >= i; j--) {
            this.data[j+1] = this.data[j];   // move elements right
        }
        this.data[i] = elem;                 // insert new element
        num++;                               // one element more!
    }
…   }
```


UNIVERSITY OF SOUTHERN DENMARK.DK

List ADT: Implementation 1

- Implementation 1 (continued):

```
public class PartialArrayListOfInt implements ListOfInt {   …
    public void remove(int i) {
         if (i < 0 || i >= num) {  throw new Index…Exception();  }
        for (int j = i; j+1 < num; j++) {
            this.data[j] = this.data[j+1];   // move elements left
        }
        num--;                                // one element less!
    }
    // DONE!
}
```


UNIVERSITY OF SOUTHERN DENMARK.DK

Dynamic Arrays

- arrays are fixed-length
- lists are variable-length
- **Idea:**
  - use an array of (fixed) length & track number of elements
  - extend array as needed by add method

add(23) add(42) add(-3) add(17) add(31)

- **Example:**

num | **5**

data

23	42	-3	17	31			

List ADT: Design & Implementation 2

- Design 2: dynamic arrays of int
- Implementation 2:

```
public class DynamicArrayListOfInt extends PartialArrayListOfInt {
    public DynamicArrayListOfInt(int limit) {
        super(limit);
    }
    public void add(int i, int elem) {
        if (i < 0 || i > num) {  throw new Index…Exception();  }
    …
}
```


UNIVERSITY OF SOUTHERN DENMARK.DK

List ADT: Implementation 2

- Implementation 2 (continued):

```
if (num >= limit) {          // array is full
    int[] newData = new int[2*this.limit];
    for (int j = 0; j < limit; j++) {  newData[j] = data[j];  }
    this.data = newData;
    this.limit *= 2;
}
for (int j = num-1; j >= i; j--) {
    this.data[j+1] = this.data[j];  // move elements right
}
this.data[i] = elem;         num++;
}   }
```

List ADT: Design 2 Revisited

- Design 2 (revisited): symmetric dynamic arrays of int
  - keep startIndex and endIndex of used indices
  - start with startIndex = endIndex = limit / 2
  - i.e., limit / 2 free positions at the beginning
  - i.e., limit / 2 free positions at the end
  - extend array at the beginning when startIndex < 0 needed
  - extend array at the end when endIndex > limit needed
  - shrink array in remove, when
    (endIndex – startIndex) < limit / 4

UNIVERSITY OF SOUTHERN DENMARK.DK

List ADT: Design 3

- goal is to use list for arbitrary data types
- Design 3: dynamic arrays of objects
- Implementation 3:

```
public class DynamicArrayList imple… List {
    private int limit;          // current maximum number
    private Object[] data;      // elements of the list
    private int num = 0;        // current number of elements
    public DynamicArrayListOfInt(int limit) {
        this.limit = limit;
        this.data = new Object[limit];
    } …
}
```

How to use with int, double etc.?

Boxing and Unboxing

- primitive types like int, double, … are not objects!

- Java provides wrapper classes Integer, Double, …
- Example:

```
Integer myInteger = new Integer(13);
int myInt = myInteger.intValue();
```

- transparent due to *automatic boxing* and *unboxing*
- Example:

```
Integer myInteger = 13;
int myInt = myInteger;
```

- useful when e.g. storing int values in a Object[]

UNIVERSITY OF SOUTHERN DENMARK.DK

List ADT: ArrayList

- Java provides pre-defined symmetric dynamic array list implementation in class java.util.ArrayList
- Example:

```
ArrayList myList = new ArrayList(10);          // initial limit 10
for (int i = 0;  i < 100;  i++) {
    myList.add(i*i);                            // list of squares of 0 … 99
}
System.out.println(myList);
for (int i = 99;  i >= 0;  i--) {
    int n = (Integer) myList.get(i);           // get returns Object
    myList.set(i, n*n);                          // now to the power of 4!
}
```

Generic Types

- type casts for accessing elements are unsafe!
- solution is to use *generic types*
- instead of using an array of objects, use array of some type E
- Example:

```
public class MyArrayList<E> implements List<E> {
    …
    private E[] data;
    …
    public E get(int i) {
        return this.data[i];
    }
}
```


UNIVERSITY OF SOUTHERN DENMARK.DK

Finding in Lists

- finding typical example for another List ADT operation
- specified by the following method signature:

```
public int indexOf(E elem) {
    for (int i = 0; i < this.size(); i++) {
        E cand = this.get(i);
        if (elem == null ? cand == null : elem.equals(cand)) {
            return i;        // found an equal element
        }
    }
    return -1;               // did not find any match
}
```

Sorting Lists

- sorting is another important List ADT operation
- many different approaches to sorting exist
- more on this: **DM507 Algorithms and Data Structures**
- Example (Selection Sort for list of int):

```
private void swap(int i1, i2) {
    E temp = this.get(i1);
    this.set(i1, this.get(i2));
    this.set(i2, temp);
}
```

num	**8**	**42**

data	**23**	**17**	**-3**	**42**	**31**	**97**	**71**	**59**

this.swap(1,3)

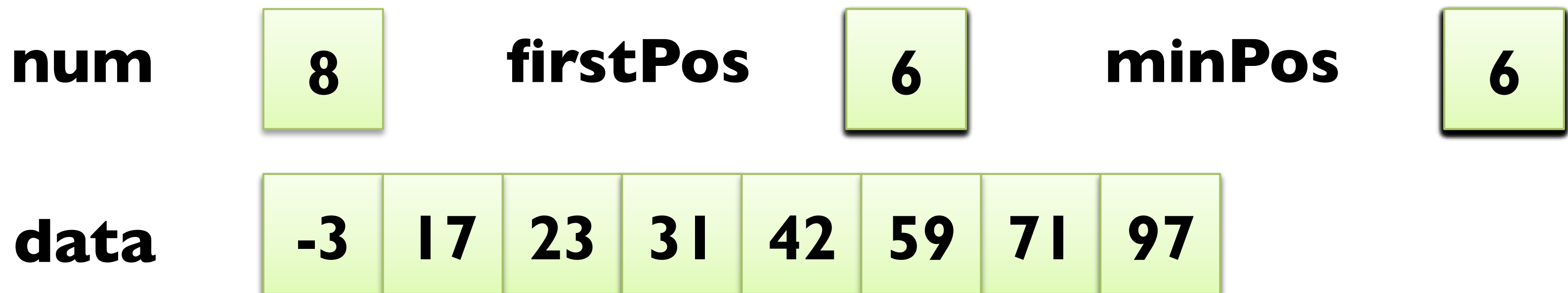
Sorting Lists

- sorting is another important List ADT operation
- many different approaches to sorting exist
- more about this in DM507 Algorithms and Data Structures
- Example (Selection Sort):

```
public void selectionSort() {
    for (int firstPos = 0;  firstPos < this.size()-1;  firstPos++) {
        int minPos = this.size()-1;  // assume last element is smallest
        for (int i = firstPos; i < this.size()-1; i++) {
            if (this.get(i) < this.get(minPos)) {  minPos = i;  }
        }
        this.swap(minPos, firstPos);
}  }
```

Sorting Lists

```
public void selectionSort() {
    for (int firstPos = 0;  firstPos < this.size()-1;  firstPos++) {
        int minPos = this.size()-1;  // assume last element is smallest
        for (int i = firstPos; i < this.size()-1; i++) {
            if (this.get(i) < this.get(minPos)) {  minPos = i;  }
        }
        this.swap(minPos, firstPos);
    }
}
```



RECURSION (REVISITED)

Recursion (Revisited)

- recursive function = a function that calls itself
- Example (meaningless):

```
public static void main(String[] args) {
    if (args.length > 0) {  main(new String[args.length-1]);  }
}
```

- base case = no recursive function call reached
- we say the function call *terminates*

- infinite recursion = no base case is reached
- also called *non-termination*

- Java recursion depth only limited by Java stack size

Comparable Interface

- imposes *total order* on elements, i.e., all elements comparable
- works similar to __cmp__(self, other) method in Python
- classes need to implement interface Comparable:

```
public interface Comparable<T> {
    public int compareTo(T other);
}
```

- has to return 0, if this and other are equal according to equals
- has to return 1, if this is greater than other
- has to return -1, if this is smaller than other

UNIVERSITY OF SOUTHERN DENMARK.DK

Binary Search

- assume ordered list of elements implementing Comparable
- "Divide et Impera" gives us *binary search* by halving the list
- Example:

```
public int find(Comparable elem) {
    return this.binarySearch(elem, 0, this.getSize()-1);
}
public int binarySearch(Comparable elem, int low, int high) {
    if (low > high) {  return -1;  }          // range is empty; not found
    int middle = (low + high) / 2;
    switch (elem.compareTo(this.get(middle))) { … }
}
```

Binary Search

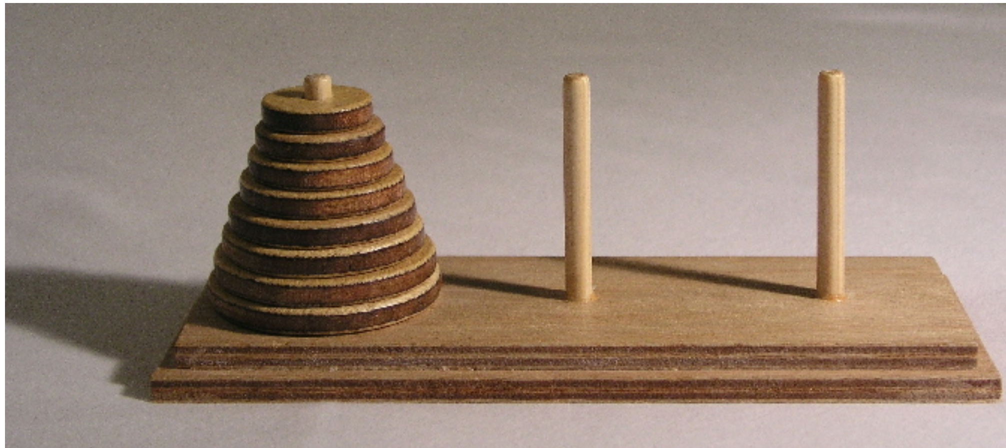
- Example (continued):

```
public int binarySearch(Comparable elem, int low, int high) {
    if (low > high) {  return -1; }        // range is empty; not found
    int middle = (low + high) / 2;
    switch (elem.compareTo(this.get(middle))) {
        case 0:    return middle;
        case 1:    return binarySearch(elem, middle+1, high);
        case -1:   return binarySearch(elem, low, middle-1);
    }
    throw new RuntimeException("compareTo error");
}
```

binarySearch(42, 4, 4)

Towers of Hanoi

- game invented by Edouard Lucas in 1883
- three pins with discs of varying diameter



- **Goal:** move all discs from the left to the right pin
- **Rule 1:** only one disc may be moved at a time
- **Rule 2:** a bigger disc may not lie on top of a smaller one
- **Rule 3:** all discs except the one moved are on some pin

Towers of Hanoi

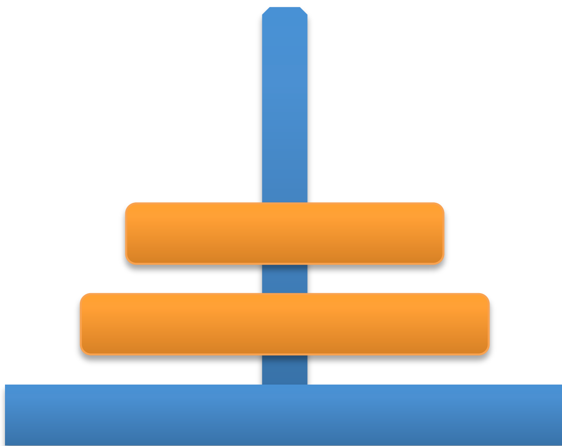
- **Solution:** move discs from left to right using the middle pin
  - move all discs (except the largest) from left pin to the middle pin using the right pin
  - move the largest disc from left to right pin
  - move all discs (except the largest) from middle pin to the right pin using the left pin

Towers of Hanoi

- **Solution:** move discs from left to right using the middle pin
  - move all discs (except the largest) from left pin to the middle pin using the right pin
  - move the largest disc from left to right pin
  - move all discs (except the largest) from middle pin to the right pin using the left pin

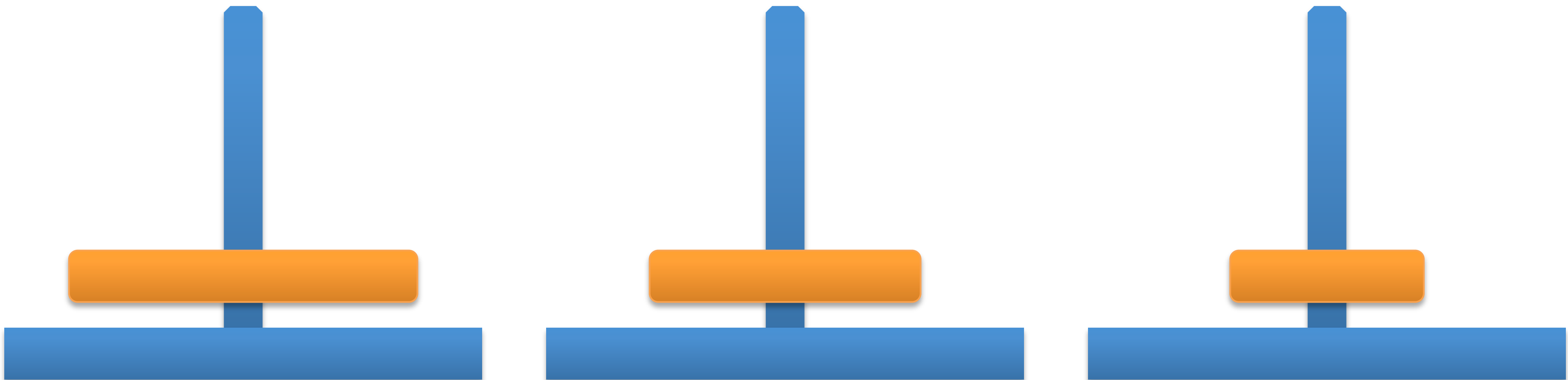
Towers of Hanoi

- **Solution:** move discs from left to right using the middle pin
  - move all discs (except the largest) from left pin to the middle pin using the right pin
  - move the largest disc from left to right pin
  - move all discs (except the largest) from middle pin to the right pin using the left pin



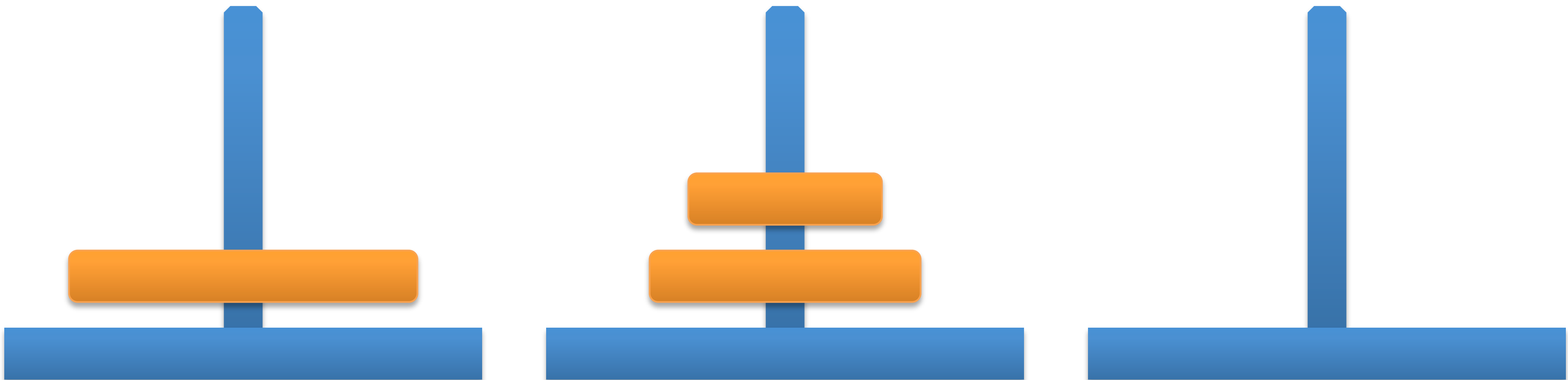
Towers of Hanoi

- **Solution:** move discs from left to right using the middle pin
  - move all discs (except the largest) from left pin to the middle pin using the right pin
  - move the largest disc from left to right pin
  - move all discs (except the largest) from middle pin to the right pin using the left pin



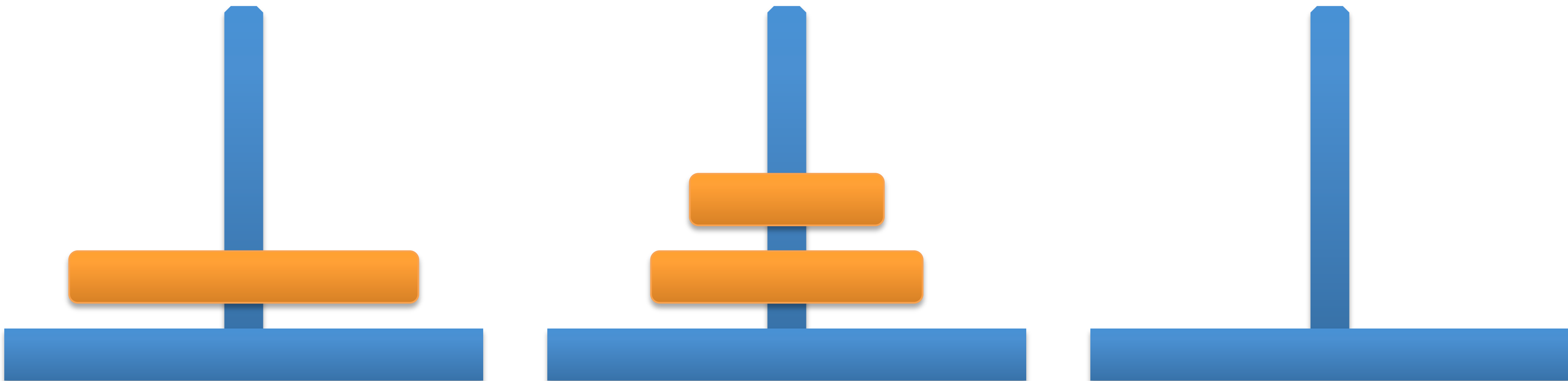
Towers of Hanoi

- **Solution:** move discs from left to right using the middle pin
  - move all discs (except the largest) from left pin to the middle pin using the right pin
  - move the largest disc from left to right pin
  - move all discs (except the largest) from middle pin to the right pin using the left pin



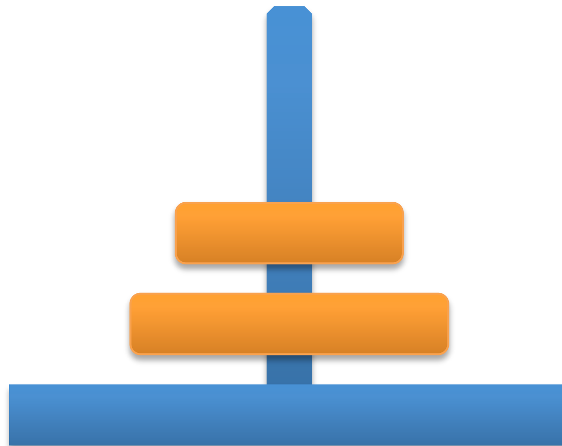
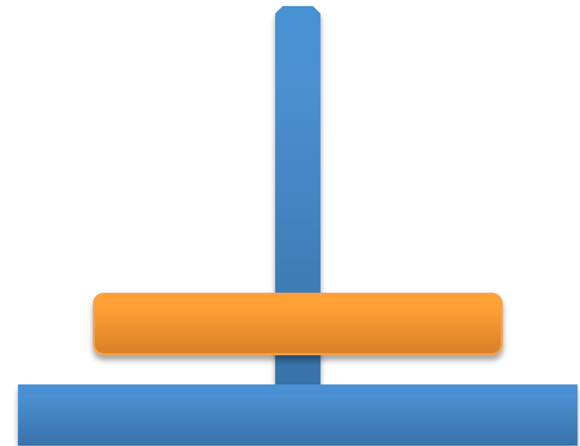
Towers of Hanoi

- **Solution:** move discs from left to right using the middle pin
  - move all discs (except the largest) from left pin to the middle pin using the right pin
  - move the largest disc from left to right pin
  - move all discs (except the largest) from middle pin to the right pin using the left pin



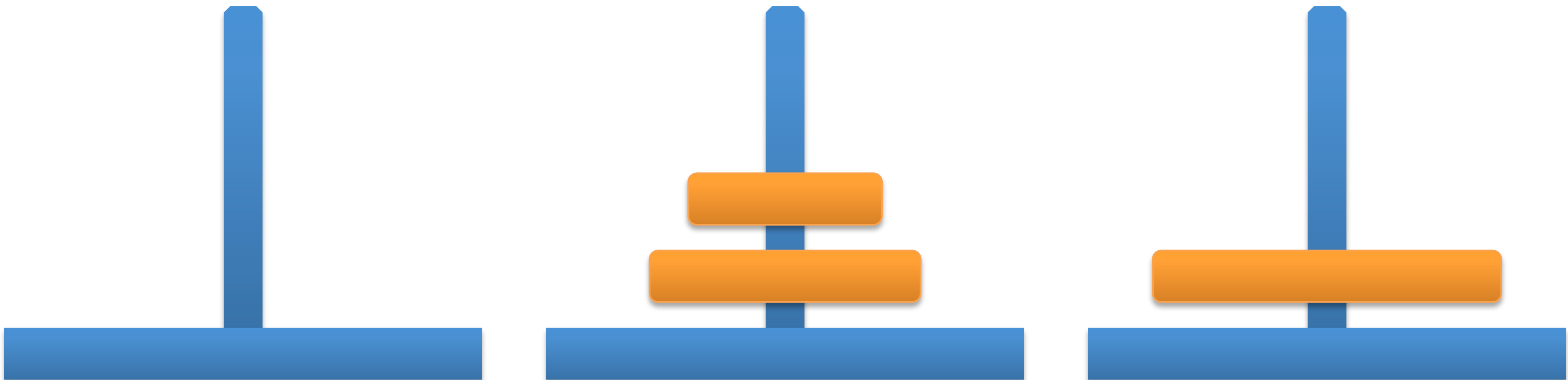
Towers of Hanoi

- **Solution:** move discs from left to right using the middle pin
  - move all discs (except the largest) from left pin to the middle pin using the right pin
  - move the largest disc from left to right pin
  - move all discs (except the largest) from middle pin to the right pin using the left pin

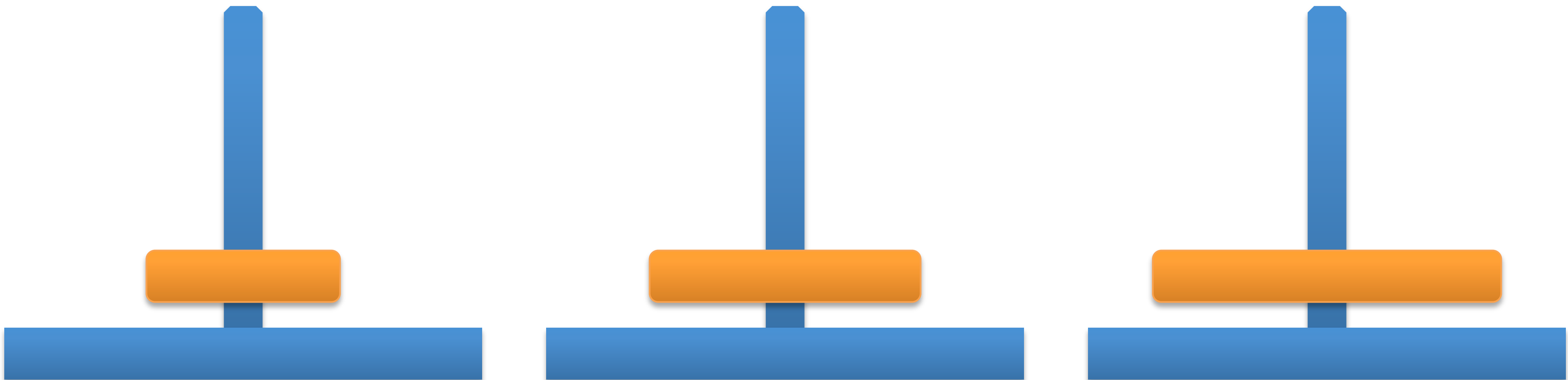
Towers of Hanoi

- **Solution:** move discs from left to right using the middle pin
  - move all discs (except the largest) from left pin to the middle pin using the right pin
  - move the largest disc from left to right pin
  - move all discs (except the largest) from middle pin to the right pin using the left pin



Towers of Hanoi

- **Solution:** move discs from left to right using the middle pin
  - move all discs (except the largest) from left pin to the middle pin using the right pin
  - move the largest disc from left to right pin
  - move all discs (except the largest) from middle pin to the right pin using the left pin



UNIVERSITY OF SOUTHERN DENMARK.DK

Towers of Hanoi

- **Solution:** move discs from left to right using the middle pin
  - move all discs (except the largest) from left pin to the middle pin using the right pin
  - move the largest disc from left to right pin
  - move all discs (except the largest) from middle pin to the right pin using the left pin

Towers of Hanoi

- **Solution:** move discs from left to right using the middle pin
  - move all discs (except the largest) from left pin to the middle pin using the right pin
  - move the largest disc from left to right pin
  - move all discs (except the largest) from middle pin to the right pin using the left pin

Towers of Hanoi

- **Solution:** move discs from left to right using the middle pin
  - move all discs (except the largest) from left pin to the middle pin using the right pin
  - move the largest disc from left to right pin
  - move all discs (except the largest) from middle pin to the right pin using the left pin
- this solution is inherently recursive
- can be formulated very simple by specifying individual moves
- to move n discs from pin 1 to pin 3 using pin 2, do:
  - move n-1 discs from pin 1 to pin 2 using pin 3
  - move disc from pin 1 to pin 3
  - move n-1 discs from pin 2 to pin 3 using pin 1

Towers of Hanoi

```
public class Move {  public int from, to;  // simply a pair of ints
    Move(int from, int to) {  this.to = to;  this.from = from;  }
    public String toString() {  return this.from+" -> "+this.to;  }
}
public static List<Move> hanoi(int n, int from, int to, int using) {
    List<Move> moves = new ArrayList<Move>();
    if (n > 1) {  moves.addAll(hanoi(n-1, from, using, to));  }
    moves.add(new Move(from, to));
    if (n > 1) {  moves.addAll(hanoi(n-1, using, to, from));  }
}
…
System.out.println(hanoi(3, 1, 3 2));
```

RECURSIVE DATA STRUCTURES

Recursive Data Structures

- like functions, data structures can be recursive, too
- recursive class = contains a member variable of same class
- Example:

```
public class Student {
    String name;
    Student tutor;
    public String toString() {
        return name + tutor == null ? "" : " (" + tutor.name + ")";
    }
}
```

- useful to implement linked lists, trees, …

UNIVERSITY OF SOUTHERN DENMARK.DK

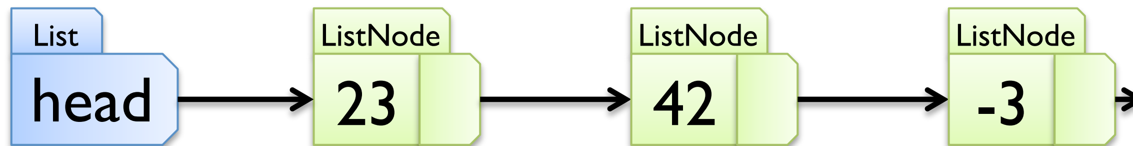
List ADT: Specification (revisited)

- data are arbitrary objects of class E
- operations are defined by the following interface

```
public interface List<E> {
    public E get(int i);                  // get i-th integer (0-based)
    public void set(int i, E elem);       // set i-th element
    public int size();                    // return length of list
    public void add(E elem);              // add element at end
    public void add(int i, E elem);       // insert element at pos. i
    public void remove(int i);            // remove i-th element
}
```

Linked Lists

- arrays require copying when inserting in the middle
- avoid copying by using links from one element to the next
- **Idea:**
  - use a recursive data structure ListNode
  - one instance of this class per element
  - every node (except the last) refers to next element



List ADT: Design & Implementation 4

- Design 4: linked lists
- Implementation 4:

```
public class ListNode<E> {
    private E elem;                           // one list element
    private ListNode<E> next;                 // next element
    public ListNode(E elem, ListNode<E> next) {
        this.elem = elem;
        this.next = next;
    }
    …
}
```


UNIVERSITY OF SOUTHERN DENMARK.DK

List ADT: Implementation 4

- Implementation 4 (continued)

```
public class ListNode<E> {
    private E elem;                    // one list element
    private ListNode<E> next;          // next element
    …
    public E get(int i) {
        if (i == 0) {  return this.elem;  }
        if (this.next == null) {  throw new Index…Exception();  }
        return this.next.get(i-1);
    }
    …
}
```


UNIVERSITY OF SOUTHERN DENMARK.DK

List ADT: Implementation 4

- Implementation 4 (continued)

```
public class ListNode<E> {
    private E elem;                          // one list element
    private ListNode<E> next;                // next element
    …
    public void set(int i, E elem) {
        if (i == 0) {  this.elem = elem;  } else {
            if (this.next == null) {  throw new Index…Exception();  }
            this.next.set(i-1, elem);
        }
    }
    …   }
```

List ADT: Implementation 4

- Implementation 4 (continued)

```
public class ListNode<E> {
    private E elem;                      // one list element
    private ListNode<E> next;            // next element
    …
    public int size() {
        int result = 1;
        if (this.next != null) {  result += this.next.size();  }
        return result;
    }
    …
}
```

List ADT: Implementation 4

- Implementation 4 (continued)

```
public class ListNode<E> {
    private E elem;                         // one list element
    private ListNode<E> next;               // next element
    …
    public void add(E elem) {
        if (this.next != null) {
            this.next.add(elem);
        } else {
            this.next = new ListNode<E>(elem, null);
        }
    }   …   }
```

List ADT: Implementation 4

- Implementation 4 (continued)

```
public void add(E elem) {
    if (this.next != null) {
        this.next.add(elem);
    } else {
        this.next = new ListNode<E>(elem, null);
    }
}
```

add(17)

List ADT: Implementation 4

- Implementation 4 (continued)

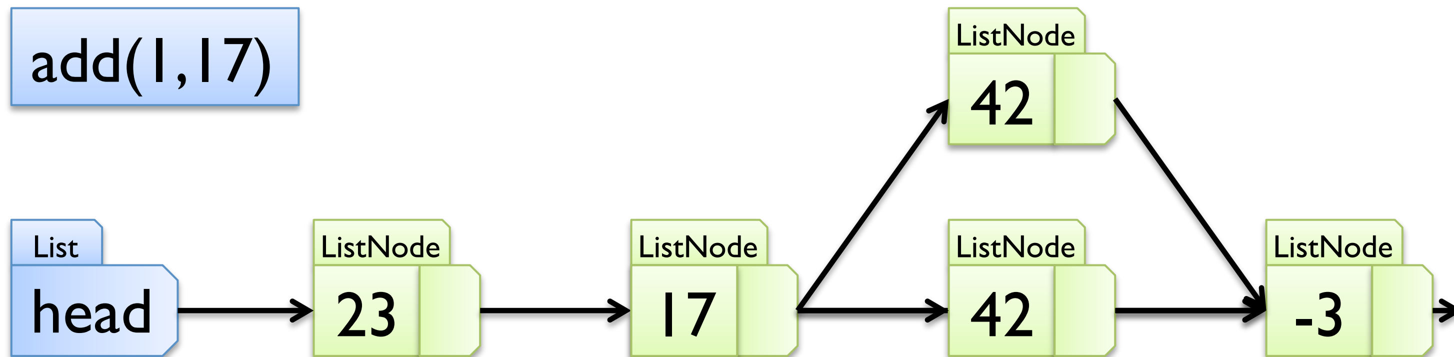
```
public void add(int i, E elem) {
    if (i == 0) {
        this.next = new ListNode<E>(this.elem, this.next);
        this.elem = elem;
    } else {
        if (this.next == null) {
            if (i == 1) {  this.add(elem);  }       // end of list
            else {  throw new Index…Exception();  }
        }  else {  this.next.add(i-1, elem);  }
    }
}
```


UNIVERSITY OF SOUTHERN DENMARK.DK

List ADT: Implementation 4

- Implementation 4 (continued)

```
public void add(int i, E elem) {
    if (i == 0) {
        this.next = new ListNode<E>(this.elem, this.next);
        this.elem = elem;
    } else {
        …
```



List ADT: Implementation 4

- Implementation 4 (continued)

```
public void remove(int i) {
    if (i == 0) {
        if (this.next == null) {  throw new RuntimeException();  }
        this.elem = this.next.elem;
        this.next = this.next.next;
    } else {
        if (this.next == null) {  throw new Index…Exception();  }
        this.next.remove(i-1);
    }
}
```

List ADT: Implementation 4

- Implementation 4 (continued)

```
public void remove(int i) {
    if (i == 0) {
        if (this.next == null) {  throw new RuntimeException();  }
        this.elem = this.next.elem;
        this.next = this.next.next;
    }   …
```

remove(1)

List ADT: Implementation 4

- Implementation 4 (continued):

```
public class LinkedList<E> implements List<E> {
    private ListNode<E> head = null;
    public E get(int i) {
        if (i < 0) {  throw new IllegalArgumentException();  }
        if (head == null) {  throw new Index…Exception();  }
        return head.get(i);
    }
    …
}
```

List ADT: Implementation 4

- Implementation 4 (continued):

```
public class LinkedList<E> implements List<E> {
    private ListNode<E> head = null;   …
    public void set(int i, E elem) {
        if (i < 0) {  throw new IllegalArgumentException();  }
        if (head == null) {  throw new Index…Exception();  }
        head.set(i, elem);
    }
    …
}
```

List ADT: Implementation 4

- Implementation 4 (continued):

```
public class LinkedList<E> implements List<E> {
    private ListNode<E> head = null;   …
    public int size() {
        if (head == null) {  return 0;  }
        return head.size();
    }
    …
}
```

List ADT: Implementation 4

- Implementation 4 (continued):

```
public class LinkedList<E> implements List<E> {
    private ListNode<E> head = null;   …
    public void add(E elem) {
        if (head == null) {
            head = new ListNode<E>(elem, null);
        } else {
            head.add(elem);
        }
    }
    …
}
```


UNIVERSITY OF SOUTHERN DENMARK.DK

List ADT: Implementation 4

- Implementation 4 (continued):

```
public class LinkedList<E> implements List<E> {   …
    public void add(int i, E elem) {
        if (i < 0) {  throw new IllegalArgumentException();  }
        if (head == null) {
            if (i > 0) {  throw new Index…Exception();  }
            head = new ListNode<E>(elem, null);
        } else {
            head.add(i, elem);
        }
    }   …
}
```

List ADT: Implementation 4

- Implementation 4 (continued):

```
public class LinkedList<E> implements List<E> {   …
    public void remove(int i) {
        if (i < 0) {  throw new IllegalArgumentException();  }
        if (head == null) {  throw new Index…Exception();  }
        else if (head.getNext() == null) {
            if (i > 0) {  throw new Index…Exception();  }
            head = null;
        } else {
            head.remove(i);
        }
    }  }
```


UNIVERSITY OF SOUTHERN DENMARK.DK