ALPEN-ADRIA UNIVERSITÄT
KLAGENFURT | WIEN GRAZ

# ESOP - Recursion / Interface / Math

Assoc. Prof. Dr. Mathias Lux
ITEC / AAU

ALPEN-ADRIA UNIVERSITÄT
KLAGENFURT | WIEN GRAZ

# Let‘s recall …

## Base Data Types

Signed, two-complement integers

- long – 64 bit
- int – 32 bit
- short – 16 bit
- byte – 8 bit

Floating point numbers

- float – 32 bit
- double – 64 bit

Others

- char - 16-bit Unicode character
- boolean – true / false

## Referecne Data Types

Everything with „new“

- Arrays
- Objects

# Wrapper Classes

- `Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`
  - wrap base data types
- Wrapper classes are reference data types
  - no base data types!
- Wrapping is partially automated
  - Autoboxing & Unboxing
- Cp. class `Boolean`

# Recursion

- A method **m()** is called *recursive*, if it calls itself.
  - m() -> m() -> m()  directly recursive
  - m() -> n() -> m()  indirectly recursive

# Recursion: Factorial n!

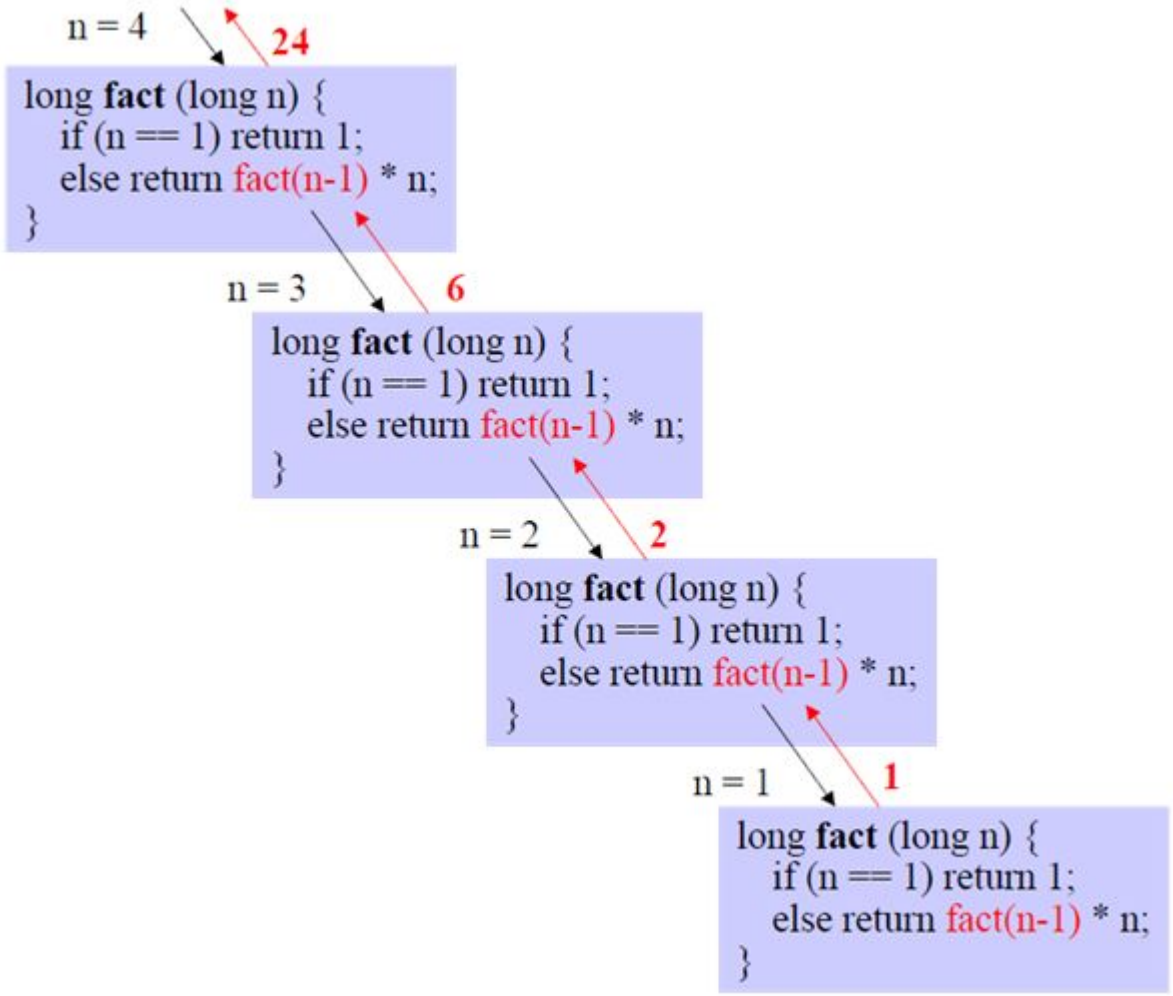
- Definition factorial
  - n! = (n-1)! * n
  - 1! = 1
- Example
  - 4! = 4*3! = 4*3*2! = 4*3*2*1! = 4*3*2*1

# Recursion: Factorial n!

```
long fact (long n) {
    if (n == 1)
        return 1;
    else
        return fact(n-1) * n;
}
```

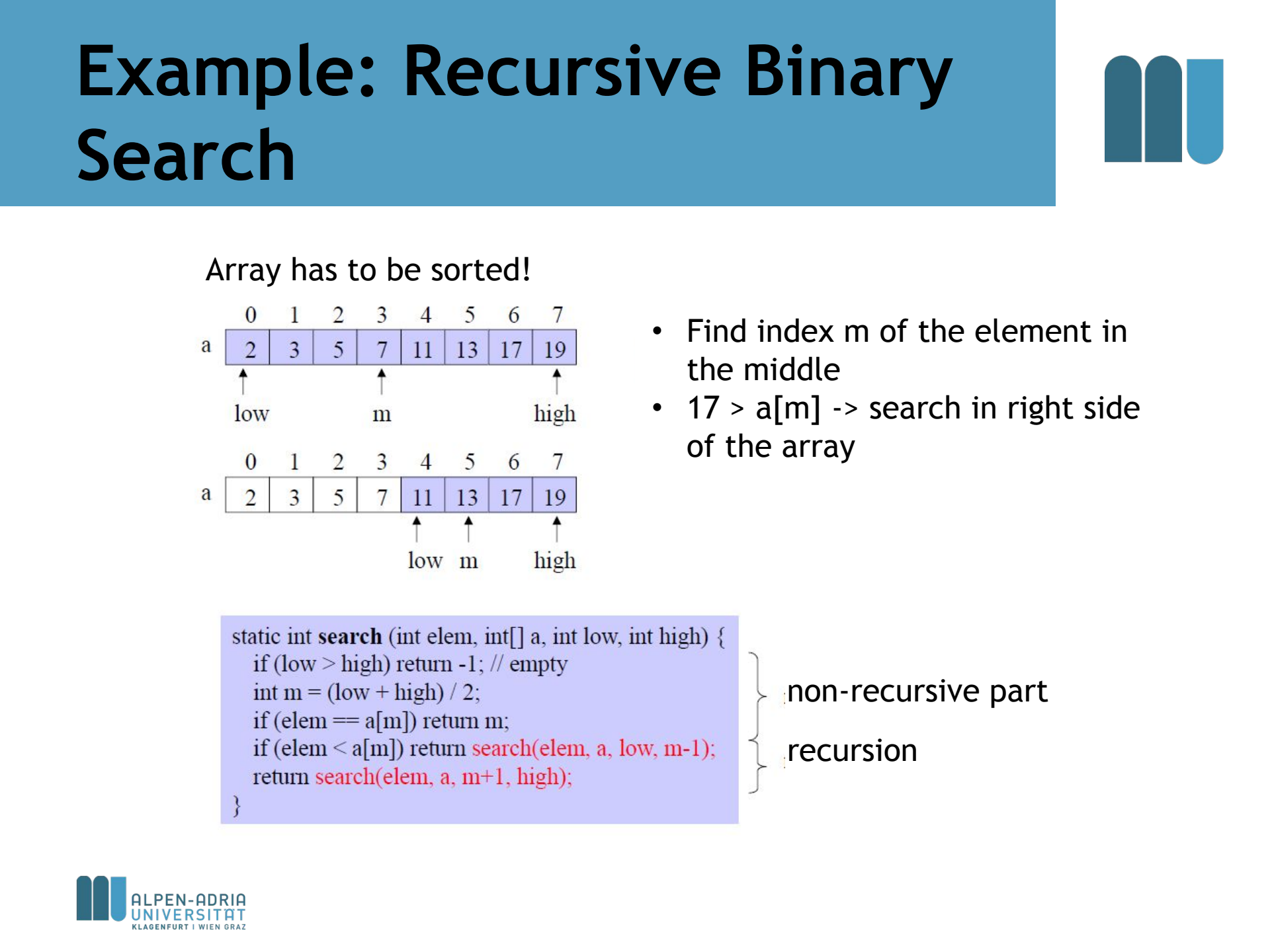
End of recursion when reaching 1!

# Recursive Process




of `fact(..)`
and stores it

# Example: Recursive Binary Search



Array has to be sorted!



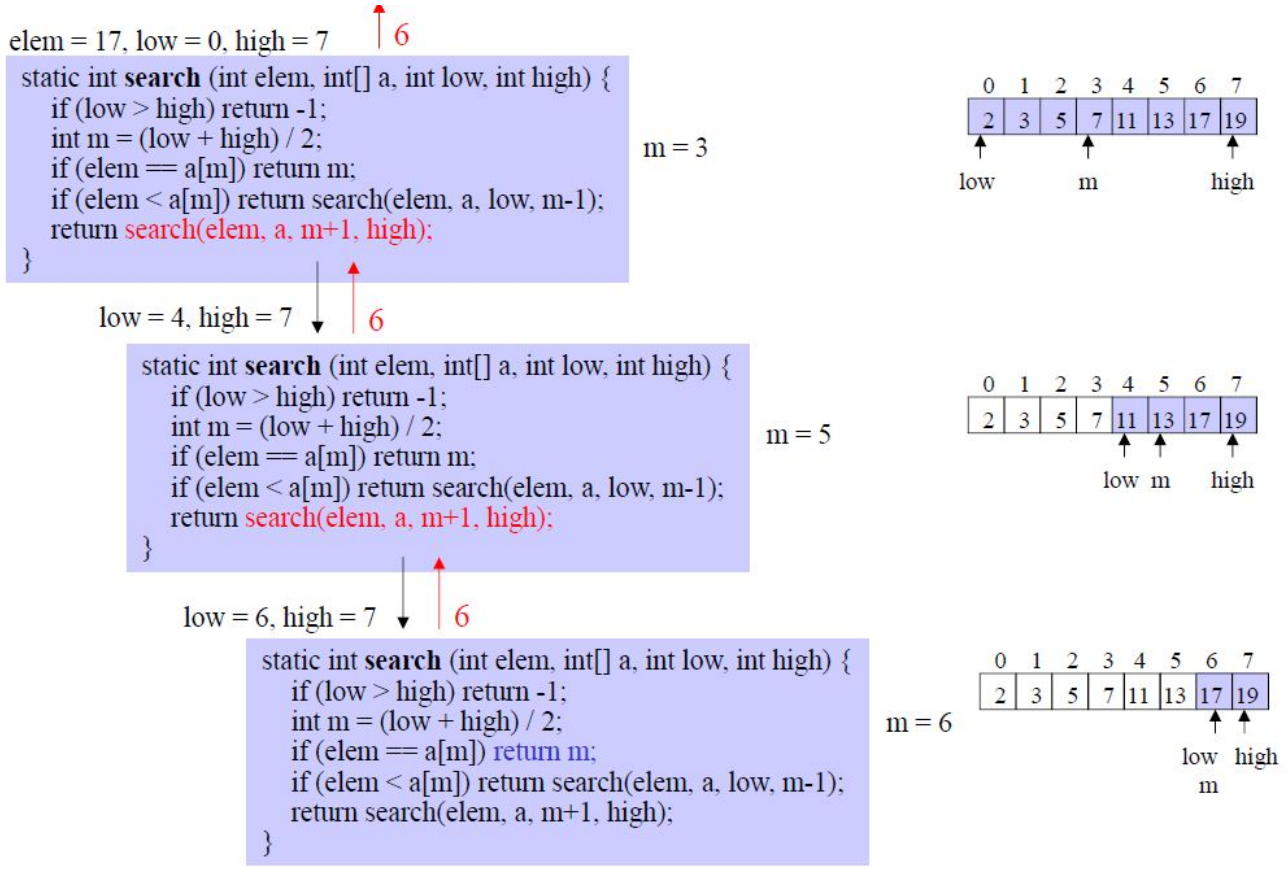

- Find index m of the element in the middle
- 17 > a[m] -> search in right side of the array

```
static int search (int elem, int[] a, int low, int high) {
   if (low > high) return -1; // empty
   int m = (low + high) / 2;
   if (elem == a[m]) return m;
   if (elem < a[m]) return search(elem, a, low, m-1);
   return search(elem, a, m+1, high);
}
```

non-recursive part

recursion

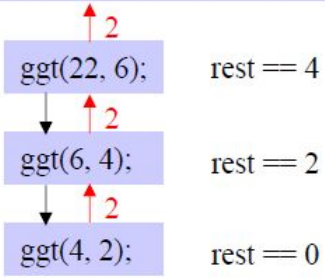
# Example: Recursive Binary Search



# Example: GCD

recursive

```
static int ggt (int x, int y) {
   int rest = x % y;
   if (rest == 0) return y;
   else return ggt(y, rest);
}
```




iterative

```
static int ggt (int x, int y) {
   int rest = x % y;
   while (rest != 0){
      x = y; y = rest;
      rest = x % y;
   }
   return y;
}
```

- Recursive algorithms can be implemented in an iterative way
  - recursive: often smaller program
  - iterative: often faster
- Recursion is extremely useful with some data structures (trees, graphs)

# Example: Fibonacci Numbers

- $F_n = F_{n-1} + F_{n-2}$

```
public static int get(int number) {
   if (number <= 2)
      return 1;
   return get(number-1) + get(number-2);
}
```

# Interfaces

- Class-like mechanism
  - for the definition of behaviour only.
- Allows for separation between definition and implementation
  - abstract data type

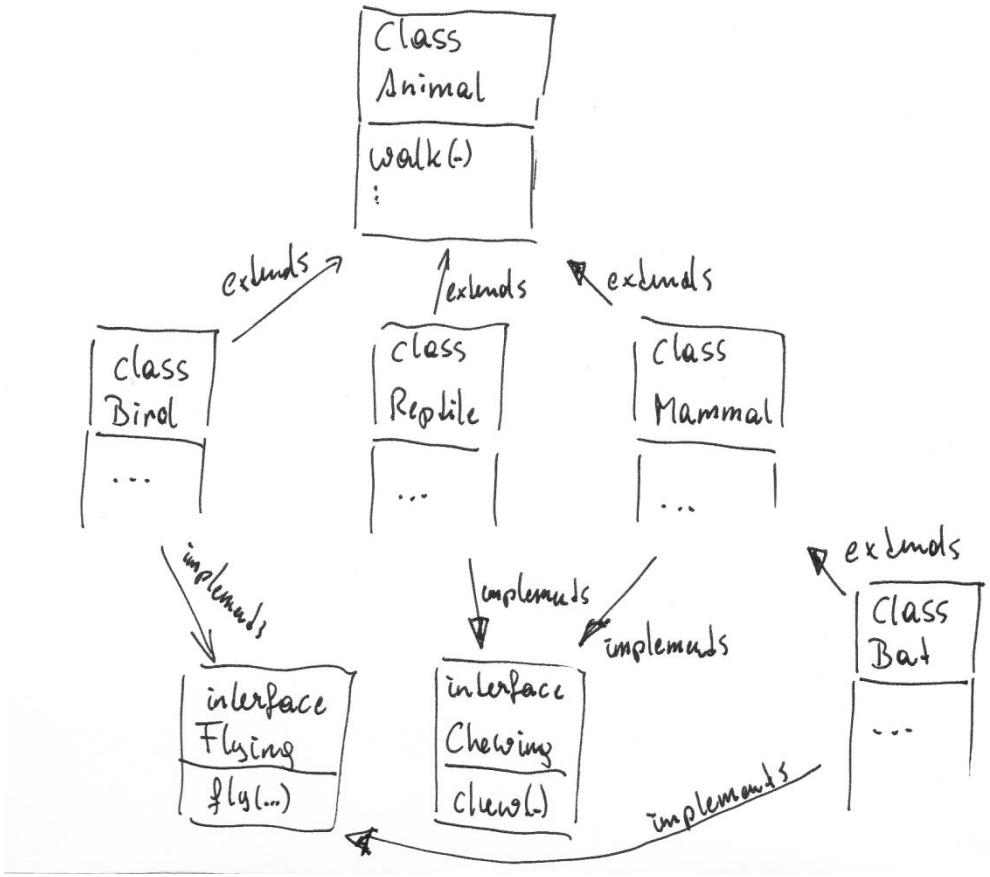
# Interfaces

- Specification via `interface` keyword
- Method specifications
  - describe how to handle the implementing object.
  - without method body, just the head
- No object variables
  - Aber evt. Konstante

# Interfaces

- The name of the interface can be used as a data type in Java.
- Implementation of an interface via `class`
  - implementing methods
  - having instance variables

# Interface Example I



# Interface Example II



java.lang

## Interface Iterable<T>

**All Known Subinterfaces:**

BeanContext, BeanContextServices, BlockingQueue<E>, Collection<E>, List<E>, Queue<E>, Set<E>, SortedSet<E>

**All Known Implementing Classes:**

AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayList, AttributeList, BeanContextServicesSupport, BeanContextSupport, ConcurrentLinkedQueue, CopyOnWriteArrayList, CopyOnWriteArraySet, DelayQueue, EnumSet, HashSet, JobStateReasons, LinkedBlockingQueue, LinkedHashSet, LinkedList, PriorityBlockingQueue, PriorityQueue, RoleList, RoleUnresolvedList, Stack, SynchronousQueue, TreeSet, Vector

```
public interface Iterable<T>
```

Implementing this interface allows an object to be the target of the "foreach" statement.

### Method Summary

| | |
|---|---|
| Iterator<T> | iterator()<br>Returns an iterator over a set of elements of type T. |

### Method Detail

**iterator**

```
Iterator<T> iterator()
```

Returns an iterator over a set of elements of type T.

# When to use Interfaces?

- Making minimal functionality of an abstract data type visible
- Multiple inheritance
  - Graph, nicht Baum

# Interface Examples

- Java Interfaces `Iterable`, `Comparable` und `Serializable`

# NameGenerator

- How can a name generator be programmed with interfaces & inheritance?