Programming in Java: lecture 9

- Eksamen

- Searching and Sorting

    - Linear and binary search

    - Insertion Sort, Selection Sort

- Multi-dimensional Arrays

    - Two dimensional arrays

- Example

Eksamen

- 48 timers hjemmeopgave, individuel

- Sidste års hjemmeopgave (en uge opgave) på hjemmesiden

- 18. juni kl 12 til 20. juni kl 12

Bestået	Ikke bestået	Syg	Ikke mødt	Antal
65	3	0	13	81

Searching

- Finding a particular element

- Linear search

- Association List

  - (key,value) pairs

  -
    ```
    class PhoneEntry {
        String name;
        String phoneNum;
    }
    ```

Linear Search

```
 * Searches the array A for the integer N.  If N is not in the array,
 * then -1 is returned.  If N is in the array, then return value is
 * the first integer i that satisfies A[i] == N.
 */
static int find(int[] A, int N) {

    for (int index = 0; index < A.length; index++) {
        if ( A[index] == N )
            return index;  // N has been found at this index!
    }

    // If we get this far, then N has not been found
    // anywhere in the array.  Return a value of -1.

    return -1;

}
```

Binary Search

- Why?

- Liniear search

  - 1000 items, max 1000 comparisons

  - 1000000 items, max 1000000 comparisons

- Binary search

  - 1000 items, 10 comparisons

  - 1000000 items, 20 comparisons

  - Data must be sorted

```
 *  Searches the array A for the integer N.
 *  Precondition:   A must be sorted into increasing order.
 *  Postcondition: If N is in the array, then the return value, i,
 *     satisfies A[i] == N.   If N is not in the array, then the
 *     return value is -1.
 */
static int binarySearch(int[] A, int N) {

    int lowestPossibleLoc = 0;
    int highestPossibleLoc = A.length - 1;

    while (highestPossibleLoc >= lowestPossibleLoc) {
        int middle = (lowestPossibleLoc + highestPossibleLoc) / 2;
        if (A[middle] == N) {
                // N has been found at this index!
            return middle;
        }
        else if (A[middle] > N) {
                // eliminate locations >= middle
            highestPossibleLoc = middle - 1;
        }
        else {
                // eliminate locations <= middle
            lowestPossibleLoc = middle + 1;
        }
    }

    // At this point, highestPossibleLoc < LowestPossibleLoc,
    // which means that N is known to be not in the array.  Return
    // a -1 to indicate that N could not be found in the array.

    return -1;

}
```
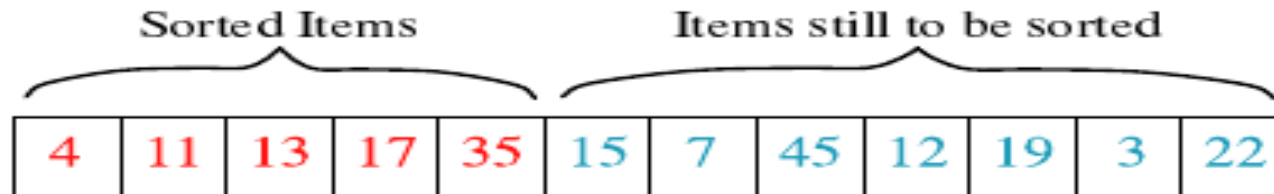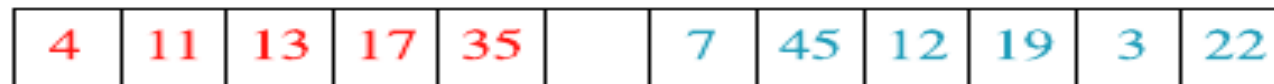
Insertion Sort

Start with a partially sorted list of items:

Sorted Items | Items still to be sorted

| 4 | 11 | 13 | 17 | 35 | 15 | 7 | 45 | 12 | 19 | 3 | 22 |

Temp: | 15 |

Copy next unsorted item into Temp, leaving a "hole" in the array.

| 4 | 11 | 13 | 17 | 35 | | 7 | 45 | 12 | 19 | 3 | 22 |

Move items in sorted part of array to make room for Temp.

Temp: | 15 |

| 4 | 11 | 13 | 15 | 17 | 35 | 7 | 45 | 12 | 19 | 3 | 22 |

Sorted Items | Items still to be sorted

| 4 | 11 | 13 | 15 | 17 | 35 | 7 | 45 | 12 | 19 | 3 | 22 |

Now, the sorted part of the list has increased in size by one item.

Insertion Sort

```
static void insertionSort(int[] A) {
    // Sort the array A into increasing order.

    int itemsSorted; // Number of items that have been sorted so far.

    for (itemsSorted = 1; itemsSorted < A.length; itemsSorted++) {

        // Assume that items A[0], A[1], ... A[itemsSorted-1]
        // have already been sorted.  Insert A[itemsSorted]
        // into the sorted part of the list.

        int temp = A[itemsSorted];  // The item to be inserted.
        int loc = itemsSorted - 1;  // Start at end of list.

        while (loc >= 0 && A[loc] > temp) {
            A[loc + 1] = A[loc]; // Bump item from A[loc] up to loc+1.
            loc = loc - 1;       // Go on to next location.
        }

        A[loc + 1] = temp; // Put temp in last vacated space.
    }
}
```

Selection Sort

```java
static void selectionSort(int[] A) {
    // Sort A into increasing order, using selection sort

  for (int lastPlace = A.length-1; lastPlace > 0; lastPlace--) {
        // Find the largest item among A[0], A[1], ...,
        // A[lastPlace], and move it into position lastPlace
        // by swapping it with the number that is currently
        // in position lastPlace.

      int maxLoc = 0;   // Location of largest item seen so far.

      for (int j = 1; j <= lastPlace; j++) {
          if (A[j] > A[maxLoc]) {
                // Since A[j] is bigger than the maximum we've seen
                // so far, j is the new location of the maximum value
                // we've seen so far.
              maxLoc = j;
          }
      }

      int temp = A[maxLoc];   // Swap largest item with A[lastPlace].
      A[maxLoc] = A[lastPlace];
      A[lastPlace] = temp;

  }  // end of for loop

}
```

Sorting

- Comparing is not always simple

```
if ( c1.getSuit() < c.getSuit() ||
        (c1.getSuit() == c.getSuit() && c1.getValue() < c.getValue()) ) {
```
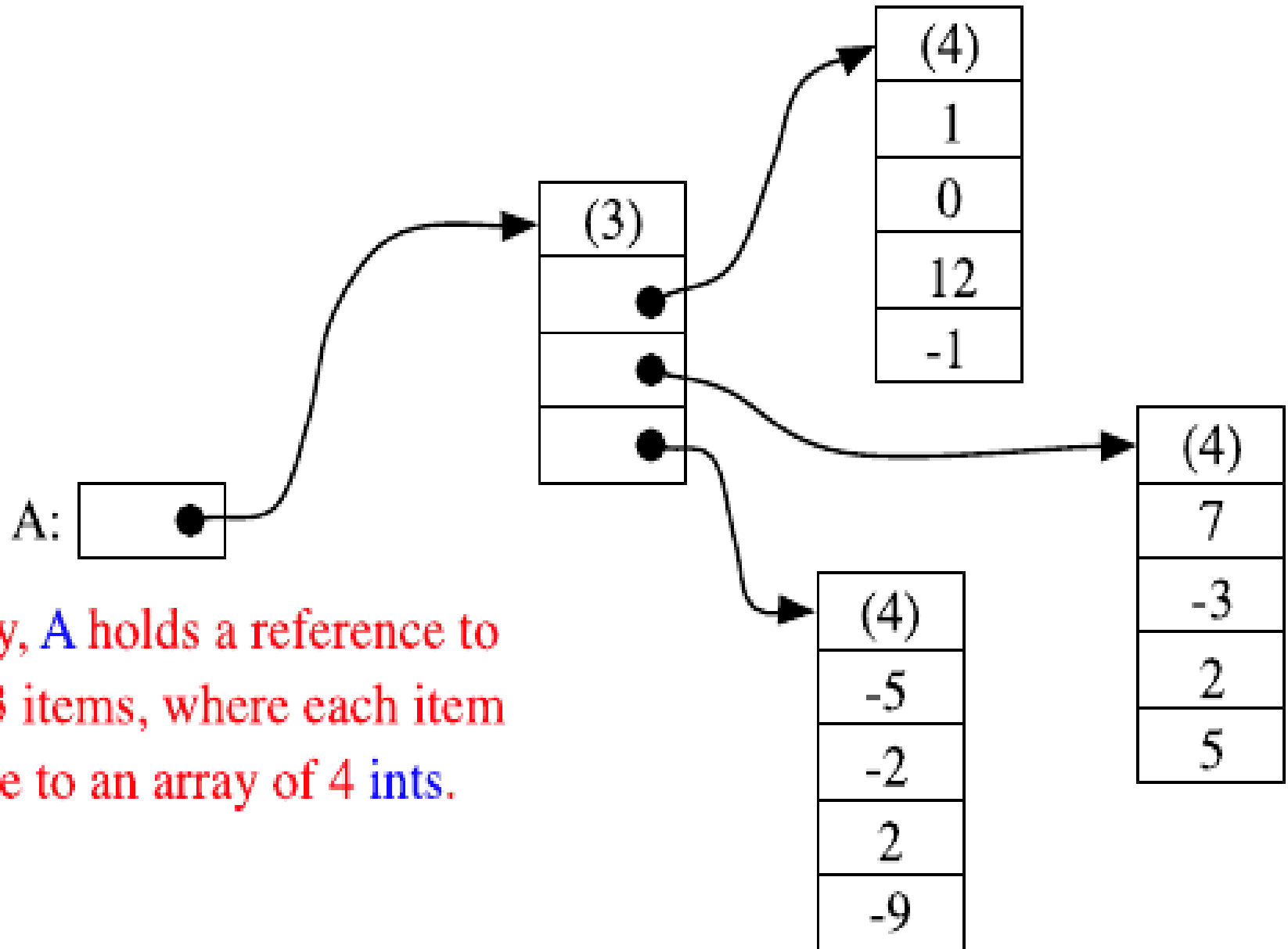
- implemented on classes

```
str1.compareTo(str2)
```

- A lot already implemented in Java

- java.util.Arrays.binarySearch()

Multi-dimensional Arrays

- int[][] A;
- A = new int[3][4];
- int[][] A = new int[3][4];

```
int [] []  A  =  { {  1,  0, 12, -1 },
                   {  7, -3,  2,  5 },
                   { -5, -2,  2, -9 }
                 };
```

Example

But in reality, A holds a reference to an array of 3 items, where each item is a reference to an array of 4 ints.

Example

A:	1	0	12	-1
	7	-3	2	5
	-5	-2	2	-9

A[0][0]   A[0][1]   A[0][2]   A[0][3]

A[1][0]   A[1][1]   A[1][2]   A[1][3]

A[2][0]   A[2][1]   A[2][2]   A[2][3]
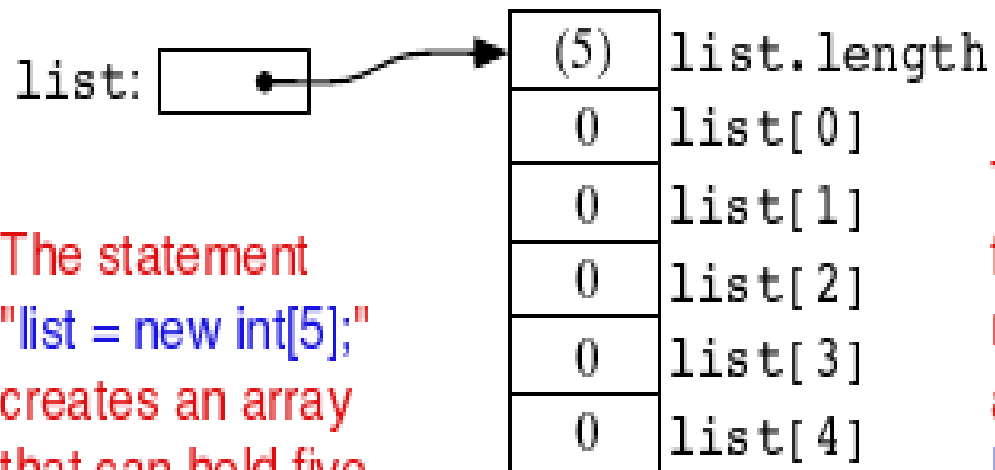
Example

- Team programming

Arrays are Objects

- int[] list; // declares the reference
- list = new int[5]; // creates an Array of size five
- list[0] = 44; // puts 44 in the first position of list

list:

(5)	list.length
0	list[0]
0	list[1]
0	list[2]
0	list[3]
0	list[4]

The statement
"list = new int[5];"
creates an array
that can hold five
ints, and sets list
to refer to it.

The array object contains
five integers, which are
referred to as list[0], list[1],
and so on. It also contains
list.length, which gives the
number of items in the array.
list.length can't be changed.

Initialization

⟨array-variable⟩ [⟨integer-expression⟩]

- Shape[] shapes = new Shape[10];
  - has null values
- double[] doubles = new double[100];
  - has 0 values
- boolean[] b = new boolean[30];
  - has default of false

Arrays Initializers

```
int[] list = { 1, 4, 9, 16, 25, 36, 49 };
```

```
Color[] palette = {
            Color.BLACK,
            Color.RED,
            Color.PINK,
            new Color(0,180,0),  // dark green
            Color.GREEN,
            Color.BLUE,
            new Color(180,180,255),  // light blue
            Color.WHITE
        };
```

Array Initializers

```
list = new int[] { 1, 8, 27, 64, 125, 216, 343 };
```

■

```
new ⟨base-type⟩ [ ] { ⟨list-of-values⟩ }
```

- In place array creation

```
makeButtons( new String[] { "Stop", "Go", "Next", "Previous" } );
```

Using Arrays

```
for (int i = 0; i < list.length; i++) {
    System.out.println( list[i] );
}
```

- ArrayIndexOutOfBoundsException

```
int count = 0;
for (int i = 0; i < A.length - 1; i++) {
    if (A[i] == A[i+1])
        count++;
}
```

Array Copy

```
double[] B = A;
```

```
double[] B = new double[A.length]; // Make a new array object,
                                    //    the same size as A.
for (int i = 0; i < A.length; i++)
    B[i] = A[i];    // Copy each item from A to B.
```

```
public static void arraycopy(Object sourceArray, int sourceStartIndex,
        Object destArray, int destStartIndex, int count)

double B = new double[A.length];
System.arraycopy( A, 0, B, 0, A.length );
```

Arrays in for-each loops

```
for ( BaseType item : anArray ) {

    .

    .   // process the item

    .

}
```

- same as

```
for ( int index = 0; index < anArray.length; index++ ) {
    BaseType item;
    item = anArray[index];   // Get one of the values from the array

        .

        .   // process the item

        .

}
```

for each loops

```
int sum = 0;    // This will be the sum of all the positive numbers in A
for ( int item : A ) {
    if (item > 0)
        sum = sum + item;
}
```

```
int[] intList = new int[10];
for ( int item : intList ) {    // INCORRECT! DOES NOT MODIFY THE ARRAY!
    item = 17;
}
```

Random Access

- You can go directly to any data value

- Other data structures where this is not the case

  - linked lists

- Inserting and removing is expensive

- Access is cheap

- Best for static size

Variable Arity Methods

```java
public static double average( double... numbers ) {
    double sum;        // The sum of all the actual parameters.
    double average;   // The average of all the actual parameters.
    sum = 0;
    for (int i = 0; i < numbers.length; i++) {
        sum = sum + numbers[i];   // Add one of the actual parameters to the sum.
    }
    average = sum / numbers.length;
    return average;
}
```