Data Abstraction
Copying Arrays. Sorting Arrays. 2D Arrays.

Janyl Jumadinova

September 30 and October 5, 2020
Copying Arrays

=: Copying only the reference - only have one actual set of contents!

System.arraycopy method:
Copy data array beginning at specified position to the specified position of
the copy array.
System.arraycopy(data, 0, copy, 0, data.length);
Copying Arrays

clone method:
Object class’s method: new array can reference the same elements as the original array.
Double[] copy = data.clone();
Copy Arrays

**Clone method:**
Object class’s method: new array can reference the same elements as the original array.
```java
Double[] copy = data.clone();
```

**Arrays.copyOf method:**
Copy contents of one array to another one
```java
Double[] copy = Arrays.copyOf(data, data.length);
```
Copying Arrays

**clone method:**
Object class’s method: new array can reference the same elements as the original array.
```java
Double[] copy = data.clone();
```

**Arrays.copyOf method:**
Copy contents of one array to another one
```java
Double[] copy = Arrays.copyOf(data, data.length);
```

These methods perform a shallow copy of the array.
Car c = new Car("Ford", 2000);

**Shallow Copy:**

The original Car object is copied exactly as it appears. All of the fields have the same values in the copy as in the original. This is why the "make" field in the copy contains the same address as in the original. A shallow copy shares all of the objects with the original beyond the first level in the object diagram.
Car c = new Car("Ford", 2000);

**Shallow Copy:**

For some Java classes shallow copy behaves the same as the deep copy because those classes are *immutable* - instance can not be changed once created (e.g., String class, wrapper classes).
Car c = new Car("Ford", 2000);

**Deep Copy:**
Original and copy are totally distinct.

```java
copy = new int[data.length];
for (int i = 0; i < data.length; i++)
    copy[i] = data[i];
```
Enhanced For Loop

double[] data = ...;
double sum = 0;
for (double element : data) {
    sum += element;
}

- Access each element in order (0 to length-1)
- Copy it to the element variable
- Execute loop body
The Sorting Problem

**Input:**
A sequence of \( n \) items, \( a_1, a_2, \ldots, a_n \)
The Sorting Problem

Input:
A sequence of $n$ items, $a_1, a_2, \ldots, a_n$

Output:
Reordering of the input sequence so that $a'_1 \leq a'_2 \leq \ldots a'_n$
Insertion Sort

Input array

5 2 4 6 1 3

At each iteration, the array is divided in two sub-arrays:

Left sub-array

2 5 4

Sorted

Right sub-array

6 1 3

Unsorted
Insertion Sort

1. 5 2 4 6 1 3
2. 2 5 4 6 1 3
3. 2 4 5 6 1 3
4. 2 4 5 6 1 3
5. 2 4 5 6 1 3
6. 1 2 4 5 6 1 3
7. 1 2 3 4 5 6 1 3
Bubble Sort

Idea:
- Repeatedly pass through the array
- Swap adjacent elements that are out of order
Bubble Sort

\[
\begin{array}{cccccccc}
8 & 4 & 6 & 9 & 2 & 3 & 1 \\
\end{array}
\]

\[
i = 1 \quad \begin{array}{cccccccc}
8 & 4 & 6 & 9 & 2 & 1 & 3 \\
\end{array}
\]

\[
i = 1 \quad \begin{array}{cccccccc}
8 & 4 & 6 & 9 & 1 & 2 & 3 \\
\end{array}
\]

\[
i = 1 \quad \begin{array}{cccccccc}
8 & 4 & 6 & 1 & 9 & 2 & 3 \\
\end{array}
\]

\[
i = 1 \quad \begin{array}{cccccccc}
8 & 4 & 1 & 6 & 9 & 2 & 3 \\
\end{array}
\]

\[
i = 1 \quad \begin{array}{cccccccc}
8 & 1 & 4 & 6 & 9 & 2 & 3 \\
\end{array}
\]

\[
i = 1 \quad \begin{array}{cccccccc}
1 & 8 & 4 & 6 & 9 & 2 & 3 \\
\end{array}
\]
Two Dimensional Arrays

- A one-dimensional array stores a list of elements.
- A two-dimensional array can be thought of as a table of elements, with rows and columns.
A two-dimensional array is an array of arrays.

A two-dimensional array is declared by specifying the size of each dimension separately:

```java
int[][] scores = new int[12][50];
```

An array element is referenced using two index values:

```java
value = scores[3][6];
```

The array stored in one row can be specified using one index.
Two Dimensional Arrays: Example

```java
int[][] table = new int[5][10];

// Load the table with values
for (int row=0; row < table.length; row++) {
    for (int col=0; col < table[row].length; col++) {
        table[row][col] = row * 10 + col;
    }
}

// Print the table
for (int row=0; row < table.length; row++) {
    for (int col=0; col < table[row].length; col++) {
        System.out.print (table[row][col] + "\t");
    }
    System.out.println();
}
```
Two Dimensional Arrays: Example

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>int[][]</td>
<td>2D array of integers, or array of integer arrays</td>
</tr>
<tr>
<td>table[5]</td>
<td>int[]</td>
<td>array of integers</td>
</tr>
<tr>
<td>table[5][12]</td>
<td>int</td>
<td>integer</td>
</tr>
</tbody>
</table>