Arrays

Arrays

- Array: Sequence of values of the same type
- Construct array:

```
new double[10]
```

- Store in variable of type `double[]`

```
double[] data = new double[10];
```

Arrays

- When array is created, all values are initialized depending on array type:
  - Numbers: 0
  - Boolean: `false`
  - Object References: `null`

Arrays

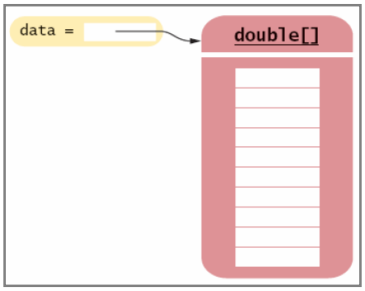

Figure 1:
An Array Reference and an Array

Arrays

- Use `[ ]` to access an element

```
data[2] = 29.95;
```

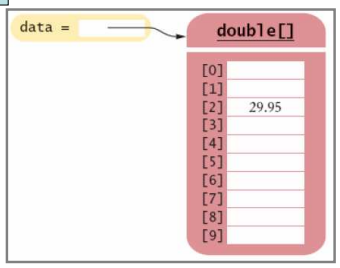

Figure 2:
Storing a Value in an Array

Arrays

- Using the value stored:

```
System.out.println("The value of this data item is " + data[4]);
```

- Get array length as `data.length`. (Not a method!)
- Index values range from `0` to `length - 1`

Arrays

- Accessing a nonexistent element results in a bounds error

```
double[] data = new double[10];
data[10] = 29.95; // ERROR
```

- Limitation: Arrays have fixed length

Syntax 8.1: Array Construction

```
new typeName[length]
```

Example:
```
new double[10]
```

Purpose:
To construct an array with a given number of elements

Syntax 8.2: Array Element Access

```
arrayReference[index]
```

Example:
```
data[2]
```

Purpose:
To access an element in an array

Self Check

1. What elements does the data array contain after the following statements?

```
double[] data = new double[10];
for (int i = 0; i < data.length; i++)
      data[i] = i * i;
```

Self Check

2. What do the following program segments print? Or, if there is an error, describe the error and specify whether it is detected at compile-time or at run-time.

```
1.  double[] a = new double[10];
    System.out.println(a[0]);
2.  double[] b = new double[10];
    System.out.println(b[10]);
3.  double[] c;
    System.out.println(c[0]);
```

Answers

1. 0, 1, 4, 9, 16, 25, 36, 49, 64, 81, but not 100
2.
   1. 0
   2. a run-time error: array index out of bounds
   3. a compile-time error: c is not initialized

Make Parallel Arrays into Arrays of Objects

```
// Don't do this
int[] accountNumbers;
double[] balances;
```

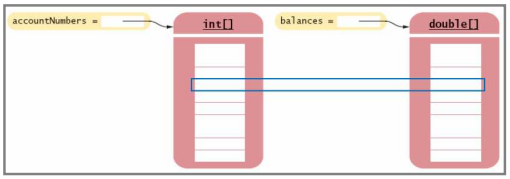

Figure 13:
Avoid Parallel Arrays

Make Parallel Arrays into Arrays of Objects

- Avoid parallel arrays by changing them into arrays of objects:

```
BankAccount[] accounts;
```

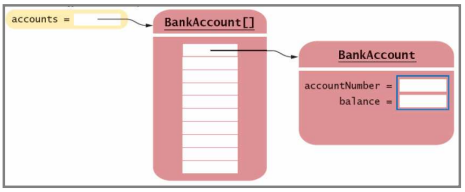

Figure 14:
Reorganizing Parallel Arrays into Arrays of Objects

Partially Filled Arrays

- Array length = maximum number of elements in array
- Sometimes, array is partially filled
- Need companion variable to keep track of current size
- Uniform naming convention:

```
final int DATA_LENGTH = 100;
double[] data = new double[DATA_LENGTH];
int dataSize = 0;
```

Partially Filled Arrays

- Update `dataSize` as array is filled:

```
data[dataSize] = x;
dataSize++;
```

Partially Filled Arrays

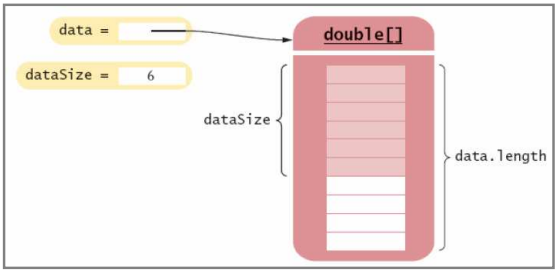

Figure 15:
A Partially Filled Array

Copying Arrays: Copying Array References

- Copying an array variable yields a second reference to the same array

```
double[] data = new double[10];
// fill array . . .
double[] prices = data;
```

Copying Arrays: Copying Array References

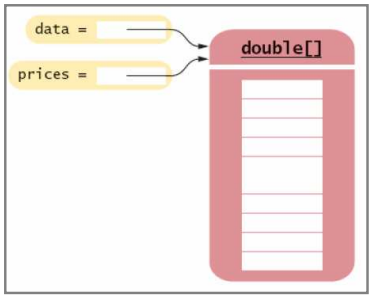

Figure 7:
Two References to the Same Array

Copying Arrays: Cloning Arrays

- Use `clone` to make true copy

```
double[] prices = (double[]) data.clone();
```

Copying Arrays: Cloning Arrays

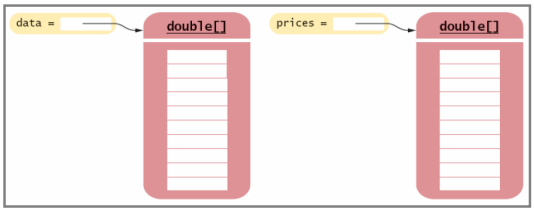

Figure 8:
Cloning an Array

Copying Arrays: Copying Array Elements

```
System.arraycopy(from, fromStart, to, toStart, count);
```

Copying Arrays: Copying Array Elements

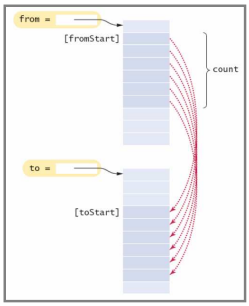

Figure 9:
The System.arraycopy Method

Adding an Element to an Array

```
System.arraycopy(data, i, data, i + 1, data.length - i - 1);
data[i] = x;
```

Adding an Element to an Array

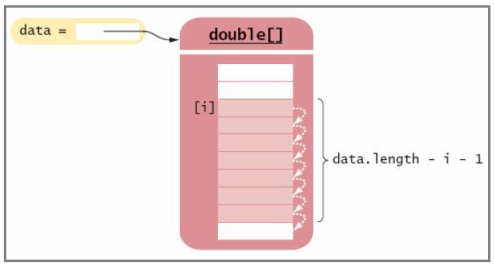

Figure 10:
Inserting a New Element Into an Array

Removing an Element from an Array

```
System.arraycopy(data, i + 1, data, i, data.length - i - 1);
```

Removing an Element from an Array

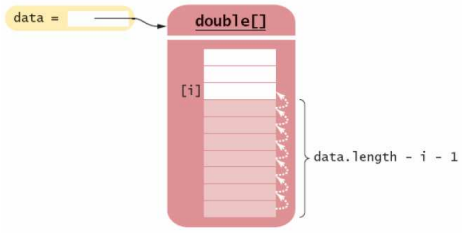

Figure 11
Removing an Element from an Array

Growing an Array

- If the array is full and you need more space, you can grow the array:

1. Create a new, larger array.

```
double[] newData = new double[2 * data.length];
```

2. Copy all elements into the new array

```
System.arraycopy(data, 0, newData, 0, data.length);
```

3. Store the reference to the new array in the array variable

```
data = newData;
```

Growing an Array

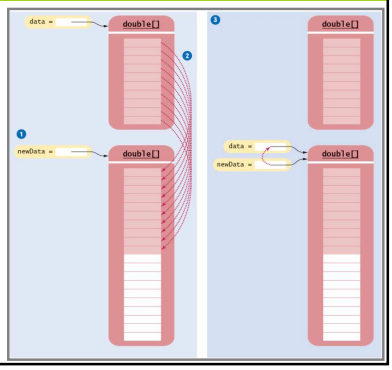

Figure 12:
Growing an Array

Question

Why do we double the length of the array when it has run out of space rather than increasing it by one element?

Answer

Allocating a new array and copying the elements is time-consuming. You wouldn't want to go through the process every time you add an element.

Two-Dimensional Arrays

- When constructing a two-dimensional array, you specify how many rows and columns you need:

```
final int ROWS = 3;
final int COLUMNS = 3;
String[][] board = new String[ROWS][COLUMNS];
```

- You access elements with an index pair `a[i][j]`

```
board[i][j] = "x";
```

A Tic-Tac-Toe Board

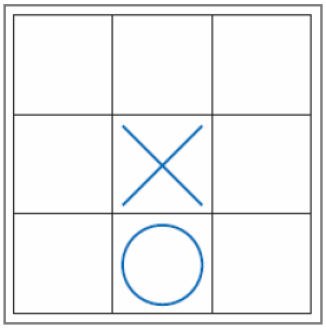

Figure 6:
A Tic-Tac-Toe Board

Traversing Two-Dimensional Arrays

- It is common to use two nested loops when filling or searching:

```
for (int i = 0; i < ROWS; i++)
   for (int j = 0; j < COLUMNS; j++)
      board[i][j] = " ";
```

Self Check

11. How do you declare and initialize a 4-by-4 array of integers?
12. How do you count the number of spaces in the tic-tac-toe board?

Answers

11.
```
int[][] array = new int[4][4];
```

12.
```
int count = 0;
for (int i = 0; i < ROWS; i++)
   for (int j = 0; j < COLUMNS; j++)
      if (board[i][j] == ' ') count++;
```