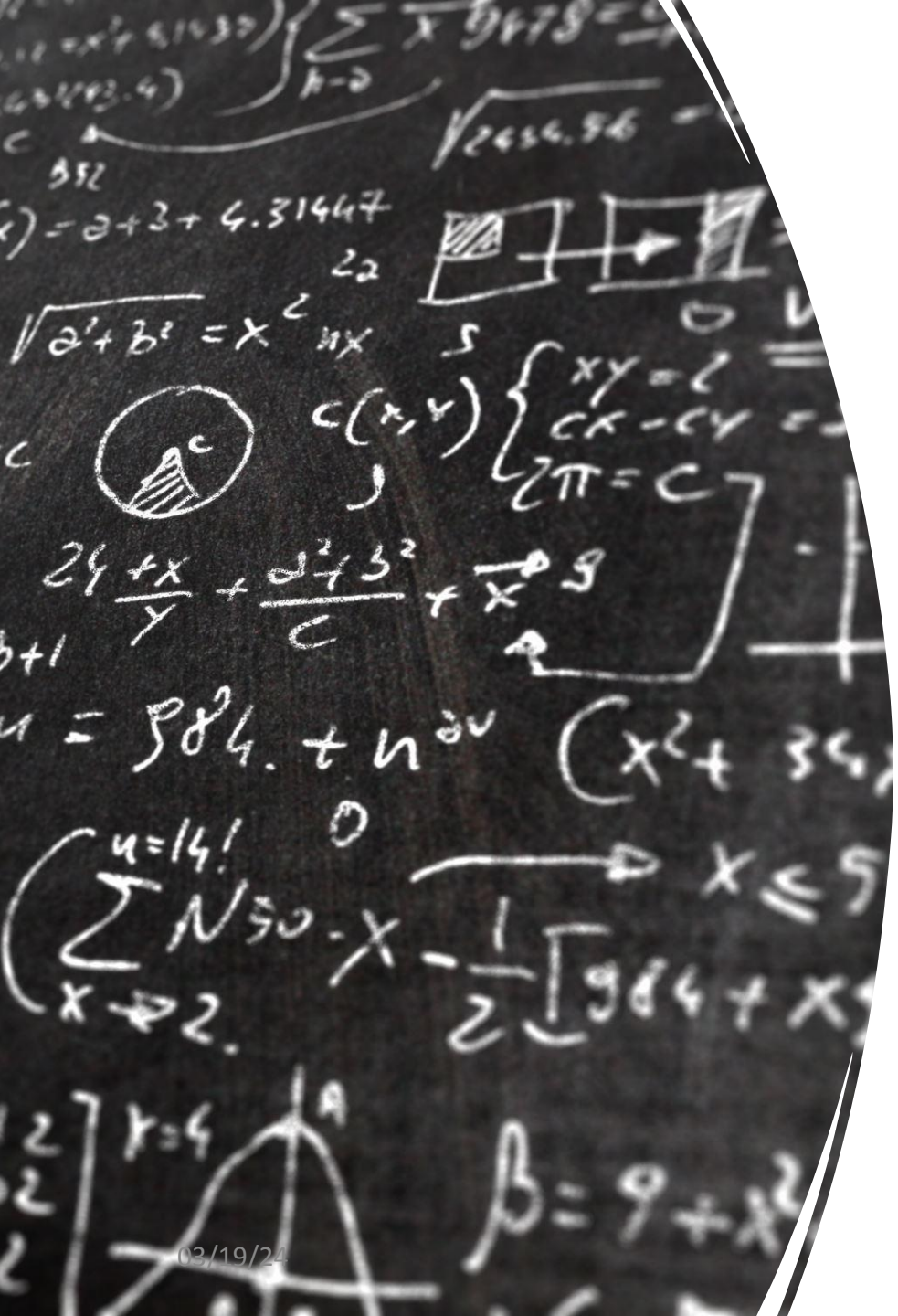


CS 113 – Computer Science I

Lecture 14 – Arrays of Arrays

Tuesday 03/19/2024

Announcements

HW 06 Loops due Sunday night

- Late due date is tonight

Midterm grades coming

Outline

1. Nested loops
2. Arrays of Arrays
3. Expanding Arrays
4. Mutability

Code Example

```
1. for (int i = 0; i <= 3; i++) {
2.     for (int j = 0; j <= 3; j++) {
3.         System.out.print(i + ", " + j + " ");
4.     }
5.     System.out.println();
6. }
```

Code Example

```
for (int i = 0; i <= 3 ; i++) {
    for (int j = 0; j <= 3; j++) {
        System.out.print(i + ", " + j + " ");
    }
    System.out.println();
}
```

i	j	i <= 3	j <= 3

Code Example

```
for (int i = 0; i <= 3 ; i++) {
    for (int j = 0; j <= 3; j++) {
        System.out.print(i + ", " + j + " ");
    }
    System.out.println();
}
```

i	j	i <= 3	j <= 3
0	0	T	T
0	1	T	T
0	2	T	T
0	3	T	T
0	4	T	F
1	0	T	T
1	1	T	T
1	2	T	T
1	3	T	T
1	4	T	F
2	0	T	T
…	…	…	….

What does this code print?

```
for (int i = 0; i < size; i++) {
    for (int j = 0; j <= i; j++) {
        System.out.print("* ");
    }
    System.out.println();
}
```

Exercise: Spelling

Write a method called canSpell that takes two strings (letters and word) and checks whether the set of letters can spell the word.

Exercise: Nested loops

```
$ java Rectangle
Enter a width: 2
Enter a height: 4
**
**
**
**

$ java Rectangle
Enter a width: 2
Enter a height: 2
**
**

$ java Rectangle
Enter a width: 7
Enter a height: 2
*******
*******
```

Arrays of Arrays

Arrays

Three ways to initialize an array

1. With an initial value

```
int[] numbers = {1, 2, 5};
```

2. With allocated space, but uninitialized

```
int[] numbers = new int[3];
```

3. With an empty array reference

```
int[] numbers = null;
```

Array Indexing

Access individual elements of an array with indexing

array[index]

We use *zero*-based indexing

first element is **0**

last element is **length-1**

Accessing indices out of range results in a **runtime error**!

Iterating through an array

Write a method called printArray that takes in an array of integers and prints out the values in the array:

printArray({1,2,3,4}) -> “1 2 3 4”

Array Comparison

we can’t use “==“ to compare arrays

Strings and arrays are **NOT** primitives

They are objects

Arrays of Arrays

int[] array1 is an array of ints

String[] array2 is an array of Strings

What is int[][] array3?

An array of integer arrays

What is String[][] array4?

An array of String arrays

2D array example

What does int[][] array = new int[4][3] look like?

2D array example

What does int[][] array = new int[4][3] look like?

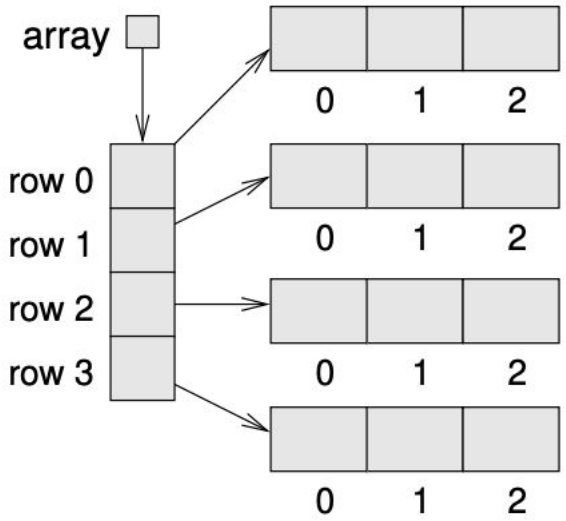


Figure 15.3: Storing rows and columns with a 2D array.

Declaring and Initializing Arrays

```
int[][] matrix1 = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
};

int[][] matrix2 = new int[3][4]; //can fill with a loop
```

2D Array

Useful for representing a:

- Grid
- Boardgame
- Matrix
- Table
- …

Looping Over a 2D array

code

Array Example

Given a 2-D array, compute the average of all elements

11	**12**	**13**	**1**	**6**
16	**17**	**18**	**9**	**8**

Array Example

write a method **fill** which takes two ints (row and col) and an int[][] and fills that position with the number 100

Array Example

Given a square 2-D array, compute the sum of the diagonal

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Array Example

Given a 2-D array, compute the sum of the perimeter

1	2	3	4	5	2	2
6	7	8	9	10	3	6
11	12	13	14	15	1	6
16	17	18	19	20	9	8

Expanding Arrays

Bank example

Keep track of account balances

Use an array:

- Each index represents another account
- The value represents the account's balance

Determine how many accounts we can hold:

- Create a new array of fixed size

Bank example

Over time our bank becomes successful, lots of new clients

No more space for new customers

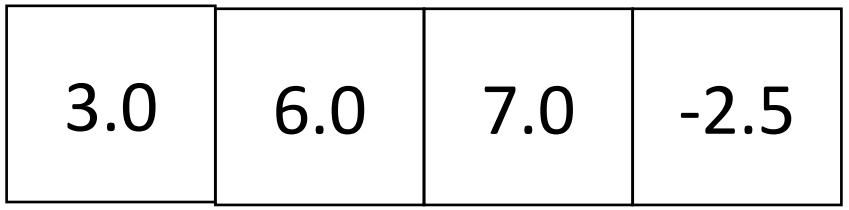
Implementation issue: running out of space in our array

Solution: build a bigger bank!

Building a bigger bank

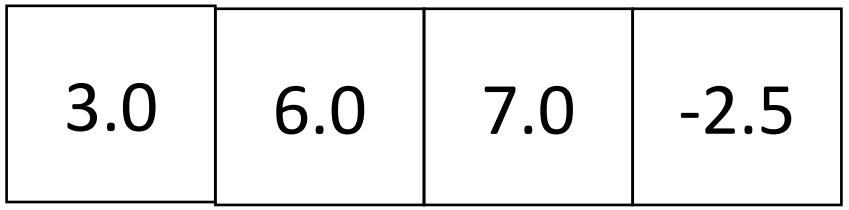
Copying arrays

Old bank

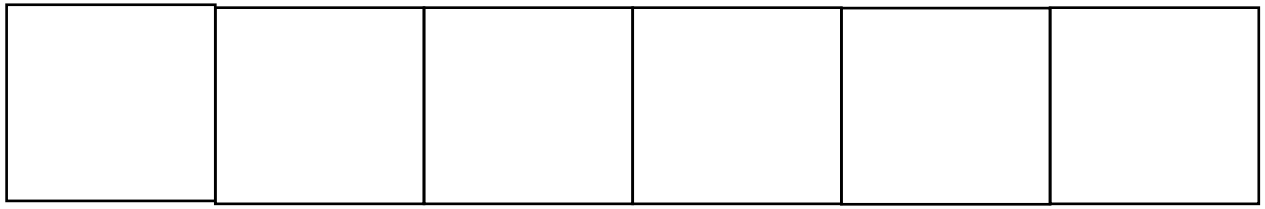


Copying arrays – build the new bank/array

Old bank

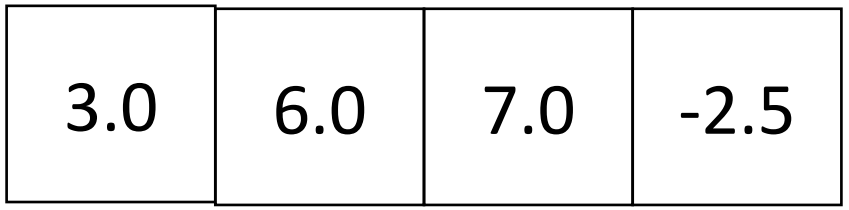


new bank

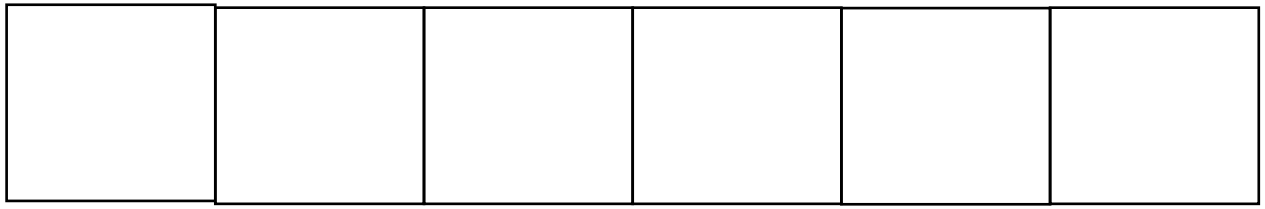


Copying arrays – copy over values/customers

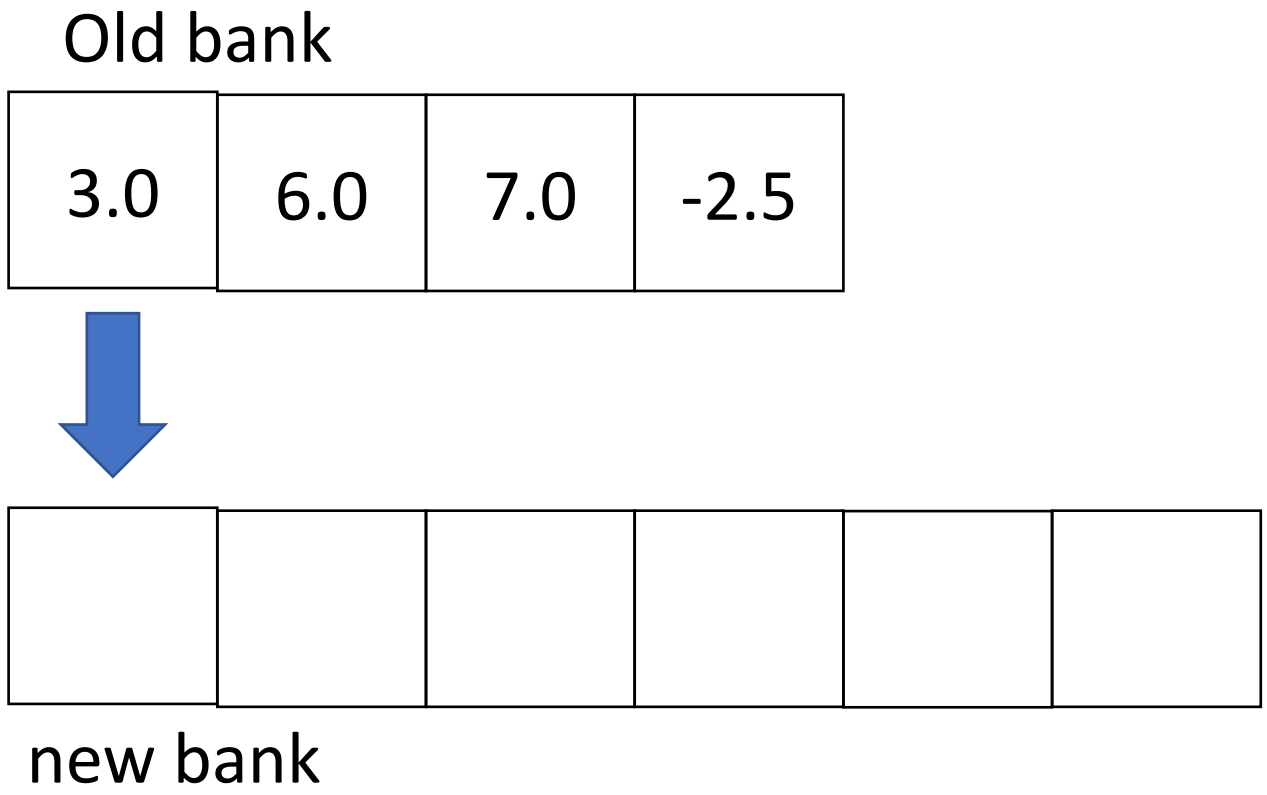
Old bank



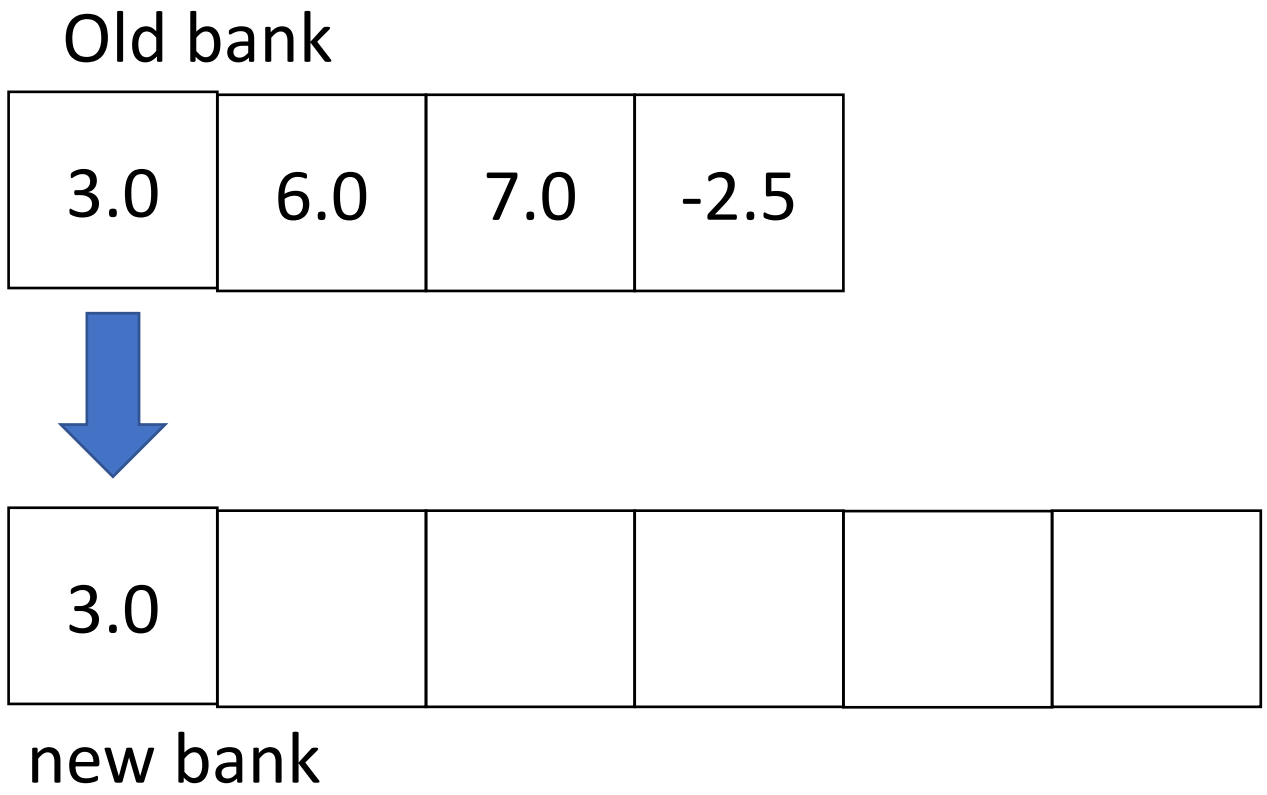
new bank



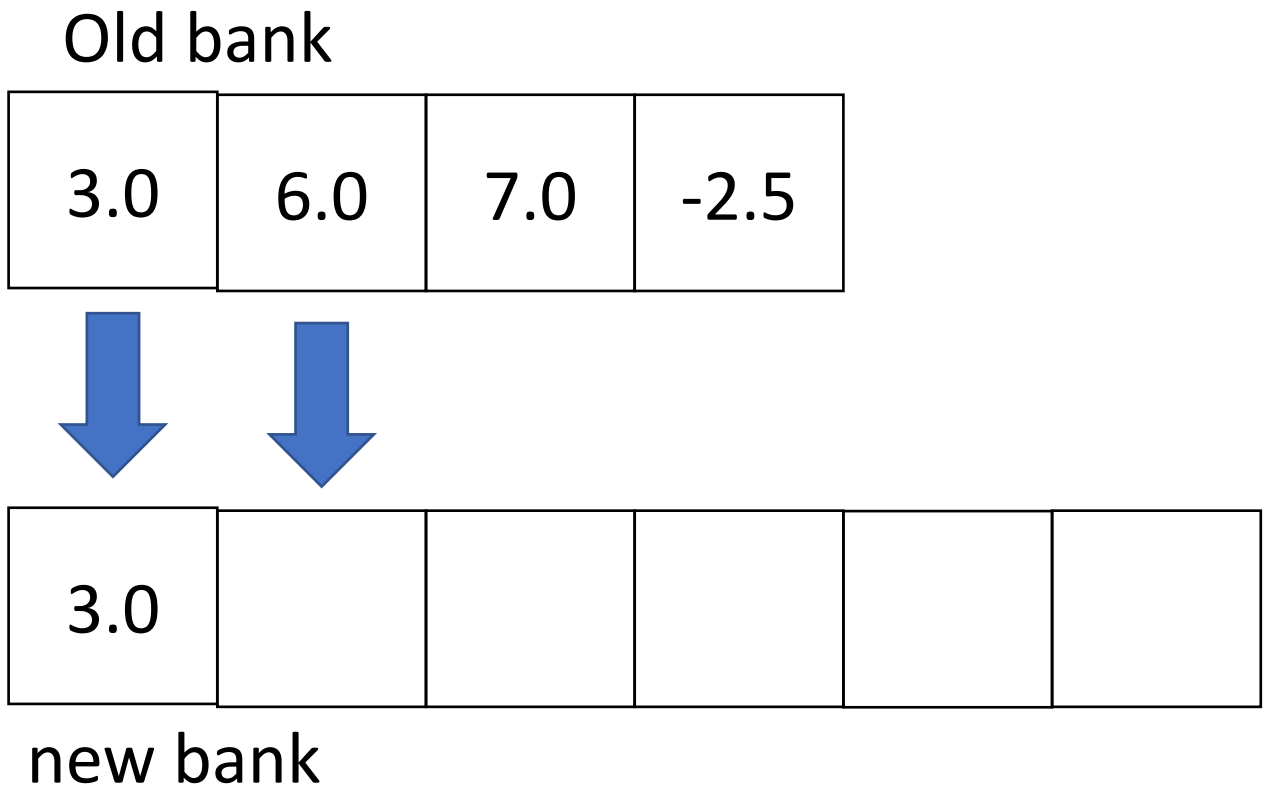
Copying arrays – copy over values/customers



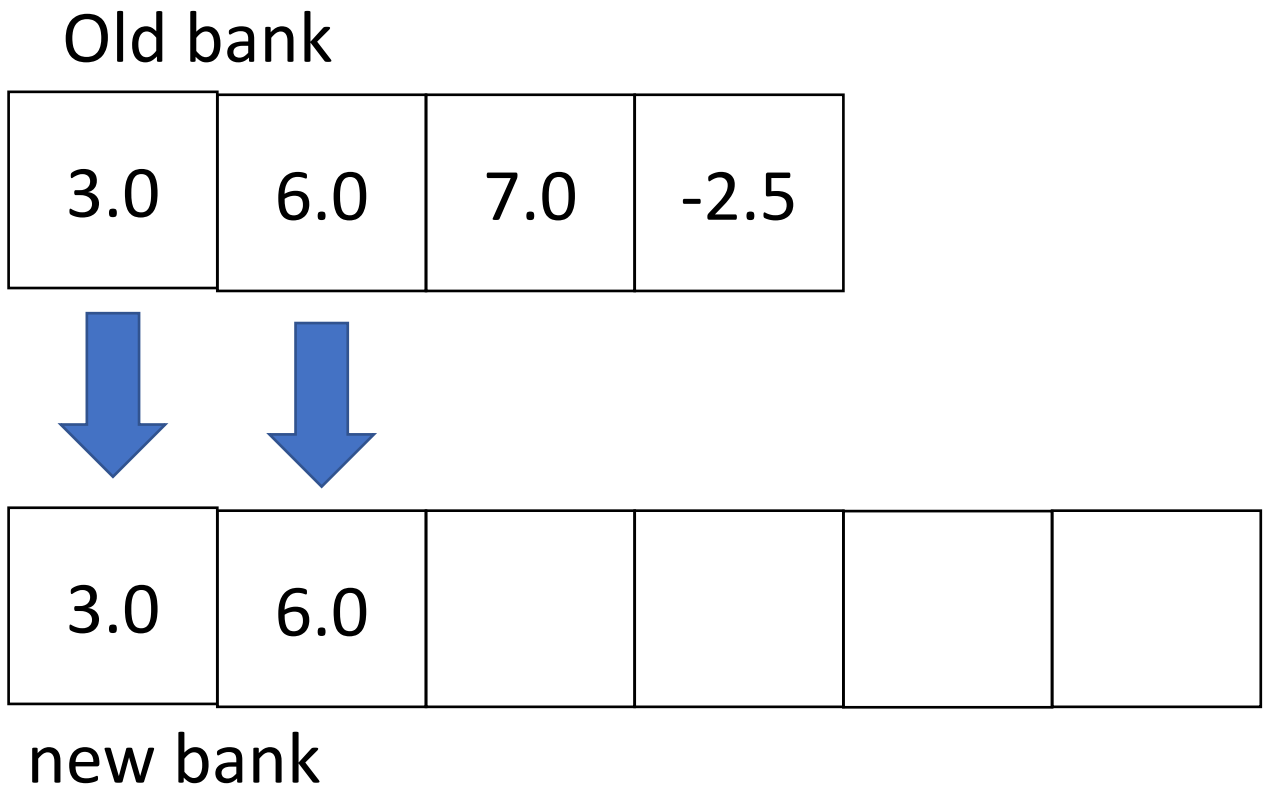
Copying arrays – copy over values/customers



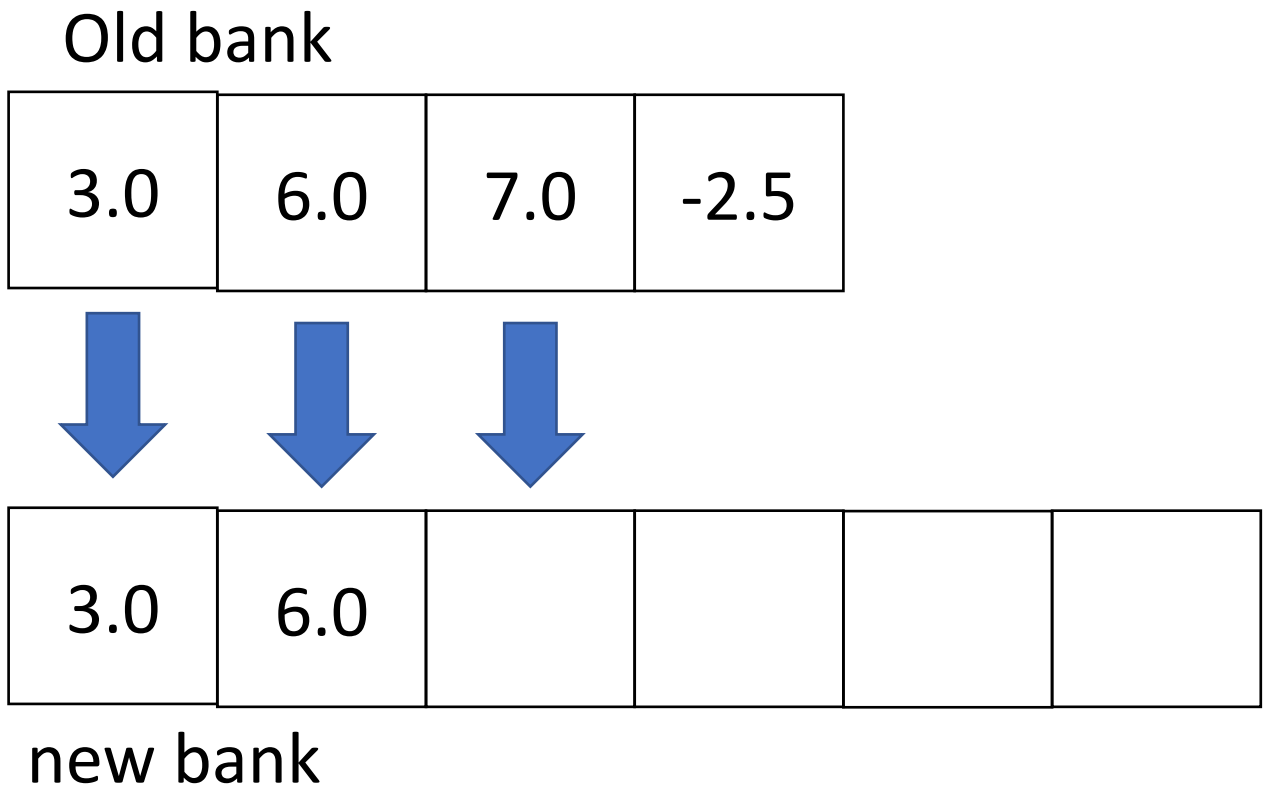
Copying arrays – copy over values/customers



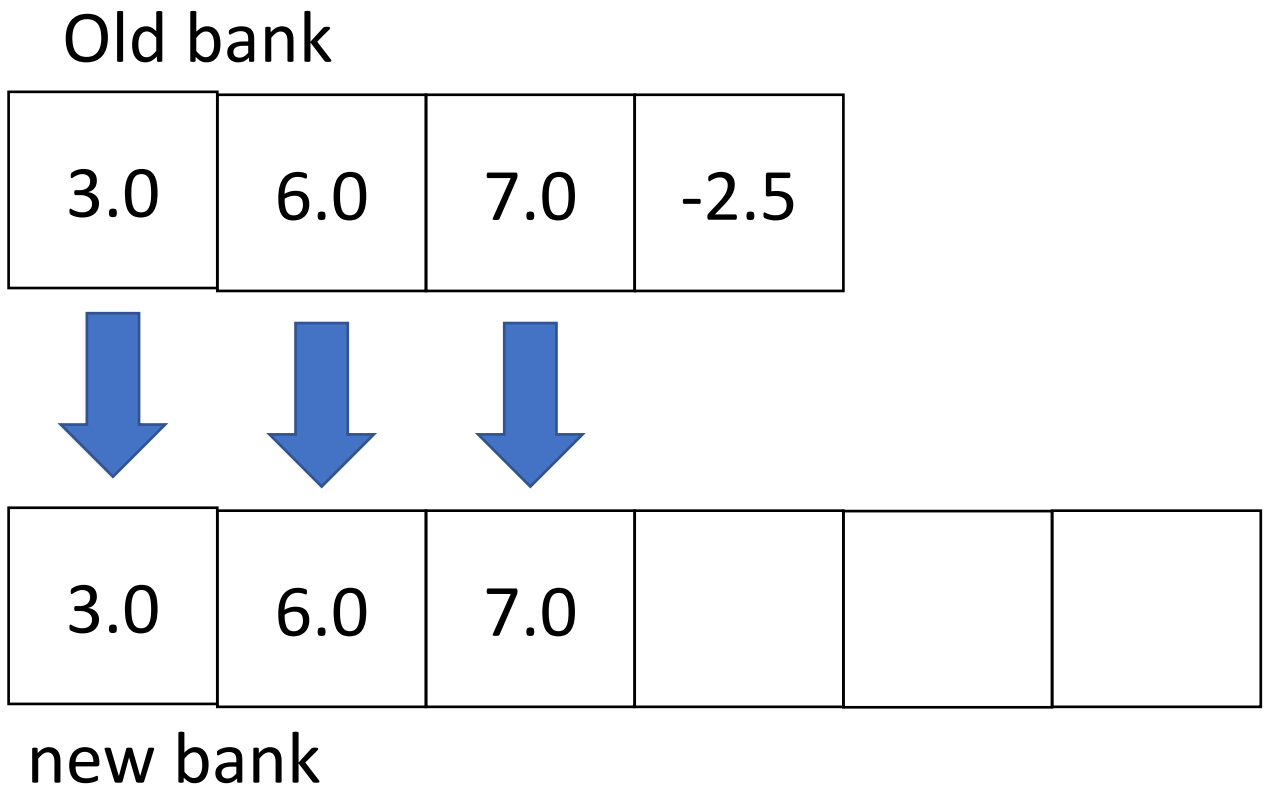
Copying arrays – copy over values/customers



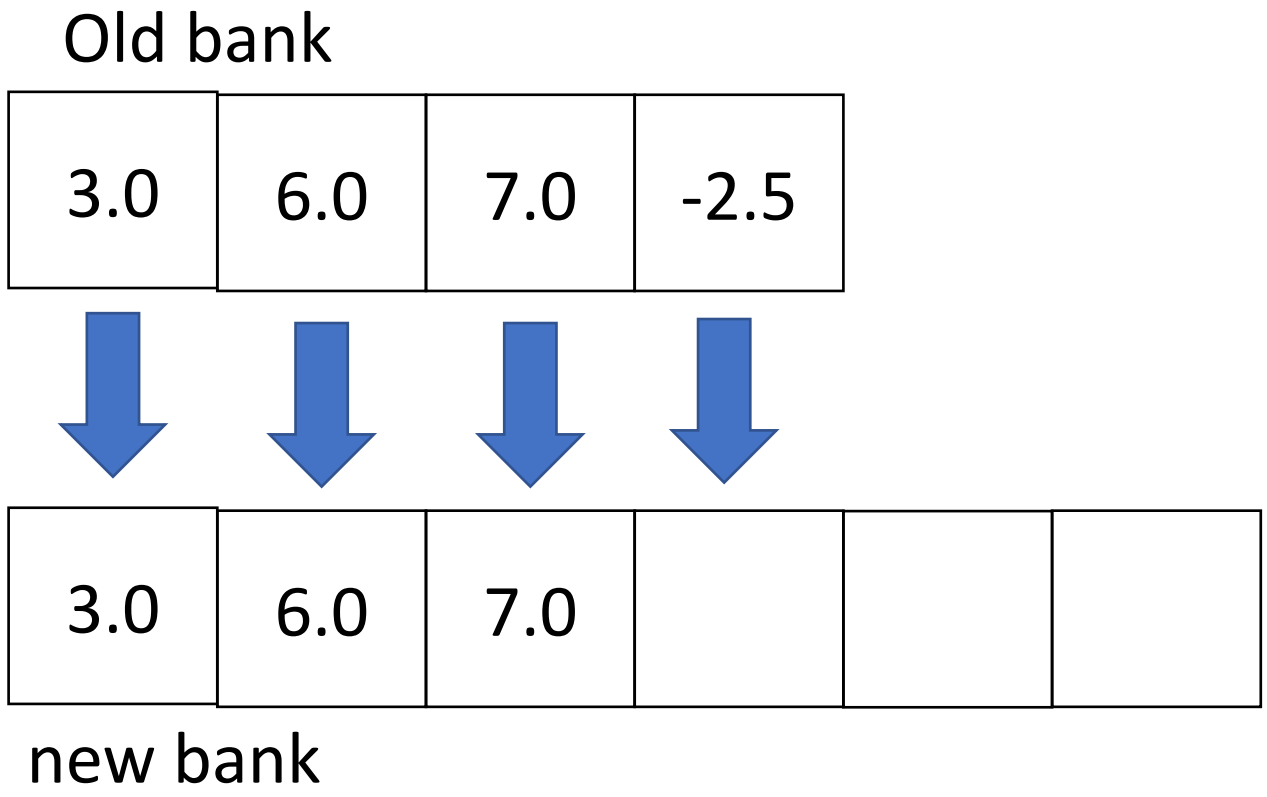
Copying arrays – copy over values/customers



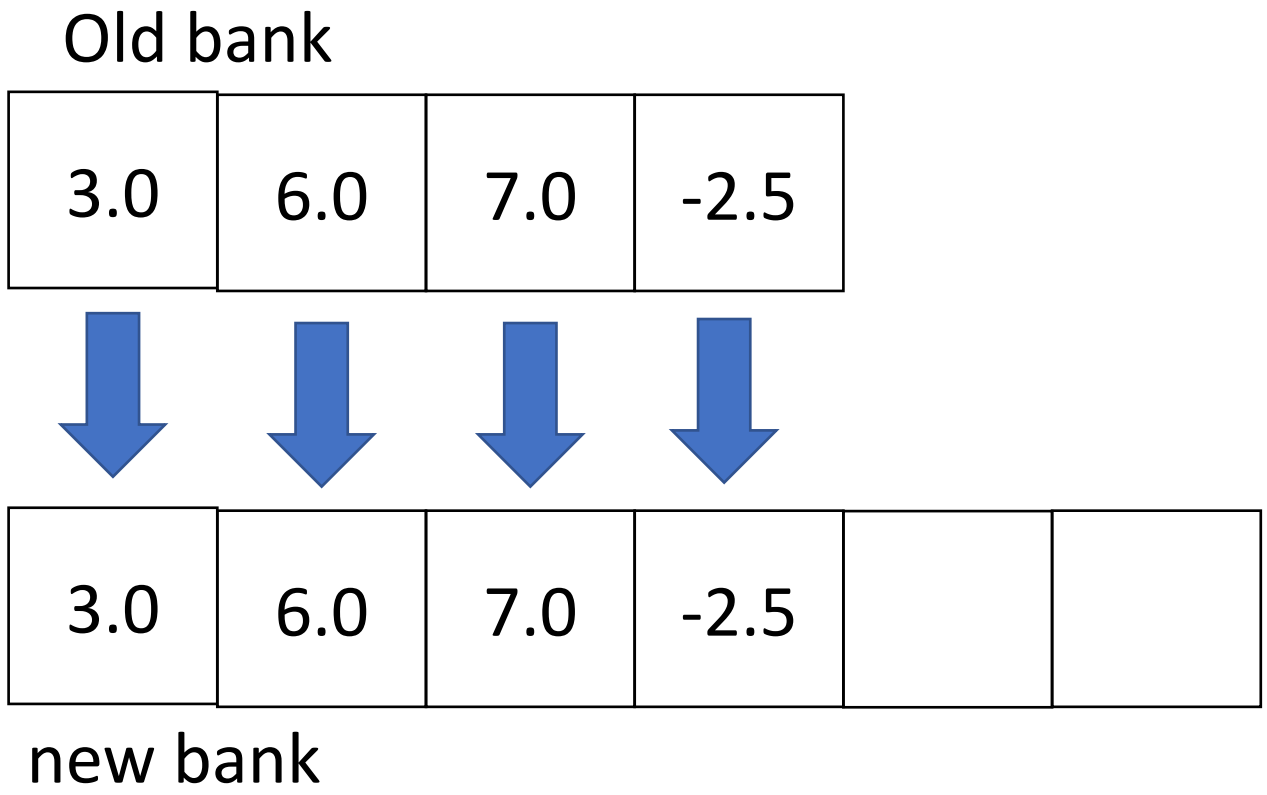
Copying arrays – copy over values/customers



Copying arrays – copy over values/customers



Copying arrays – copy over values/customers



Algorithm

When we run out of space in an array

- Create a new array (that's a bit bigger)
- Copy over all elements from the older array to the new array

Let's implement this..

How many steps do we take in this algorithm?

- Creating a new array – 1 step
- Copying *n* elements from the old array to the new array – *n* steps

How big should the new array be?

Previous size plus 1

- Pro: not making too much space
- Con: might have to create new arrays a lot of times

As big as possible

- Pro: rarely have to create a new array
- Con: wasted space

Typical solution – previous size x 2

Mutability

Mutable vs Immutable

Mutable:

- Values can change
- methods can change the state of the object directly

Immutable:

- Values cannot change
- Instead, any operations that would alter the object's state return a new object with the modified state

Strings and Integers are immutable

Mutable vs Immutable

claim: **Strings and Integers are immutable**

code!

The underlying value can change, but this will create a **new object**

Mutability

Why does java enforce this?

- Safety
  - Compile time checks force you to reason about what can be changed, what objects are being created, etc.
- Predictability
  - Any enforcements on how code will behave is a great thing for finding bugs automatically!
- Resource Management
  - Compiler can make optimizations on space needed for each object etc.