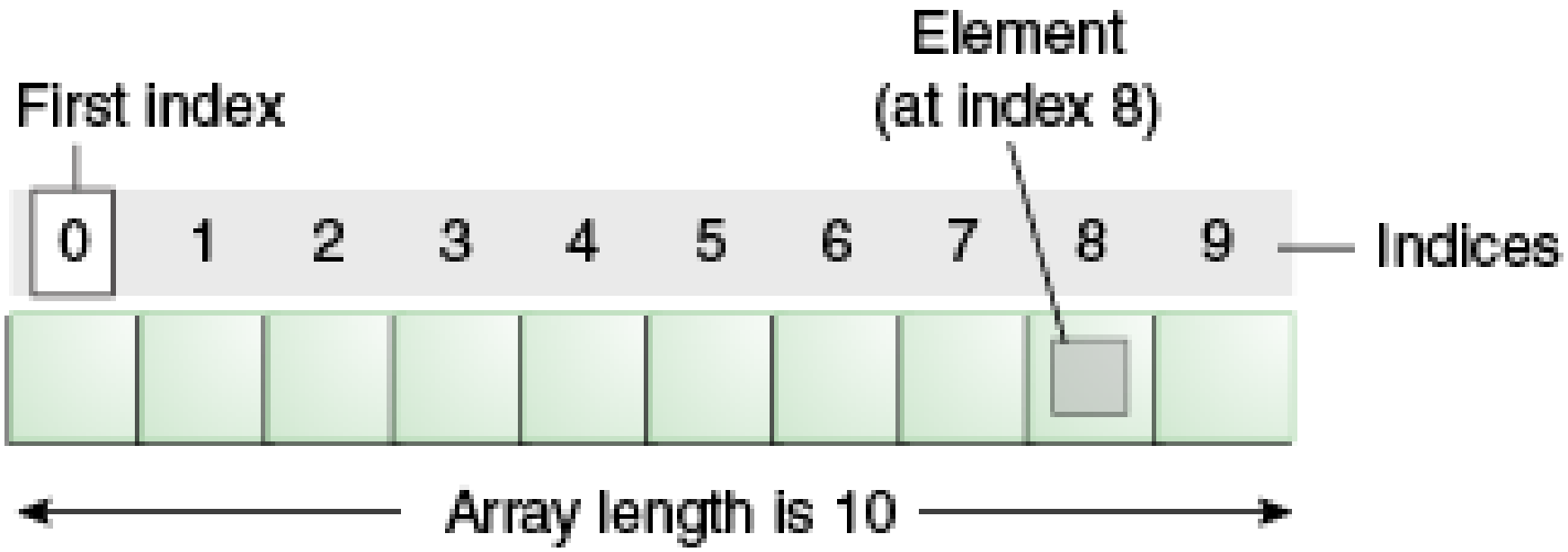
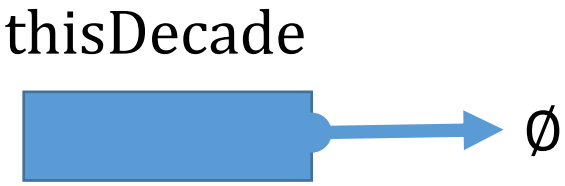
Chap. 7

Java Arrays



Arrays are "built-in" Objects

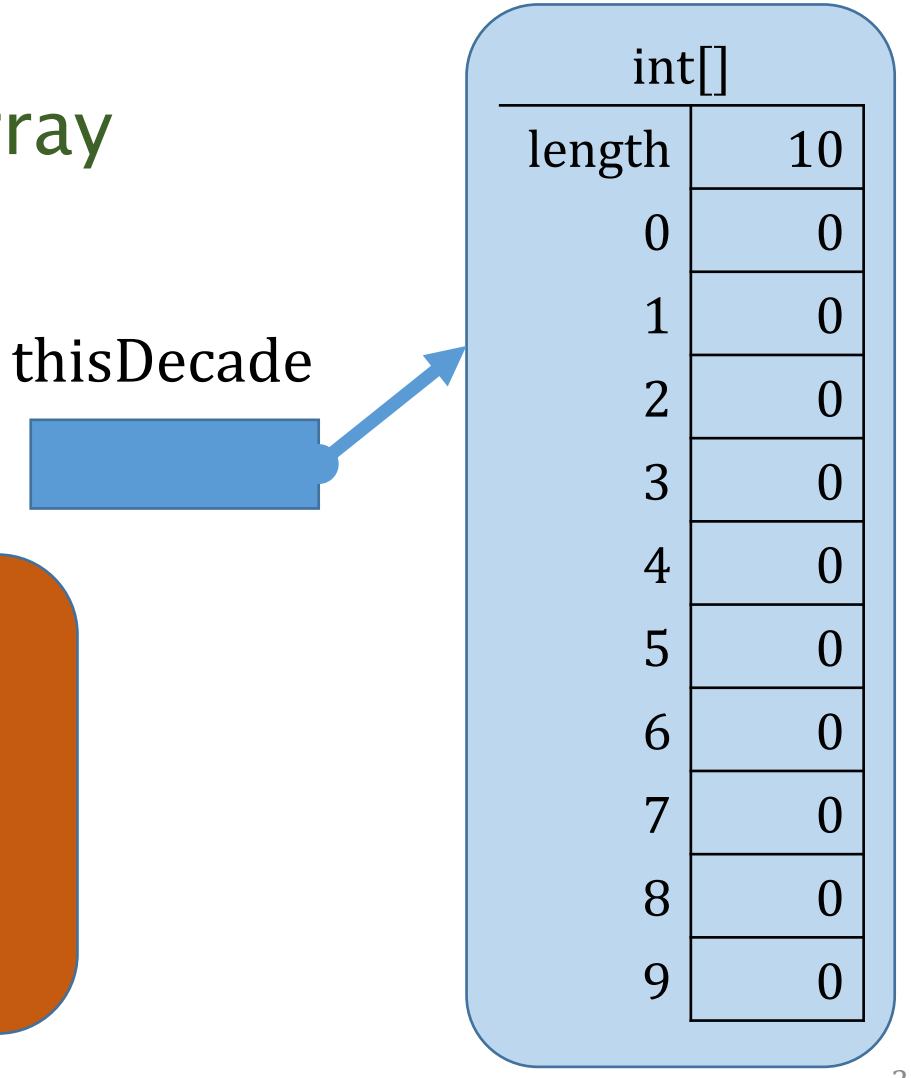
```
int[ ] thisDecade; // Reference to an array
```



thisDecade is a reference to an array of integers, but the array has not been instantiated.

Arrays are Objects

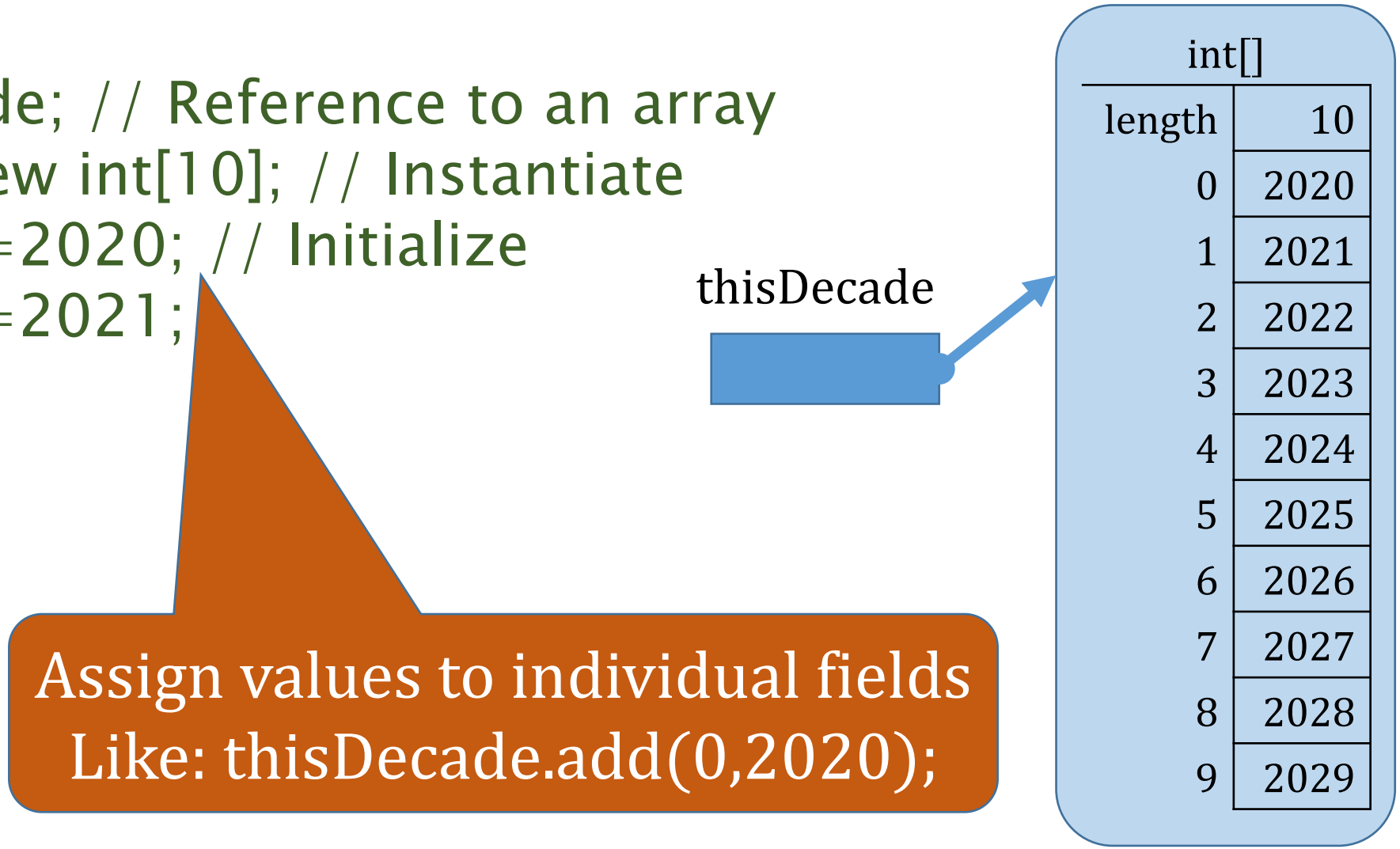
```
int[ ] thisDecade; // Reference to an array
thisDecade=new int[10]; // Instantiate
```



Create a new array object of length 10 with int fields initialized to zero
Like: new Array<int>(10)

Arrays are Objects

```
int[ ] thisDecade; // Reference to an array
thisDecade=new int[10]; // Instantiate
thisDecade[0]=2020; // Initialize
thisDecade[1]=2021;
…
```



Arrays as Objects

- Once an array is created, you cannot change its size!
- Fields in the object…
  - length – the number of items in this array
  - 0
  - 1
  - …
  - length-1

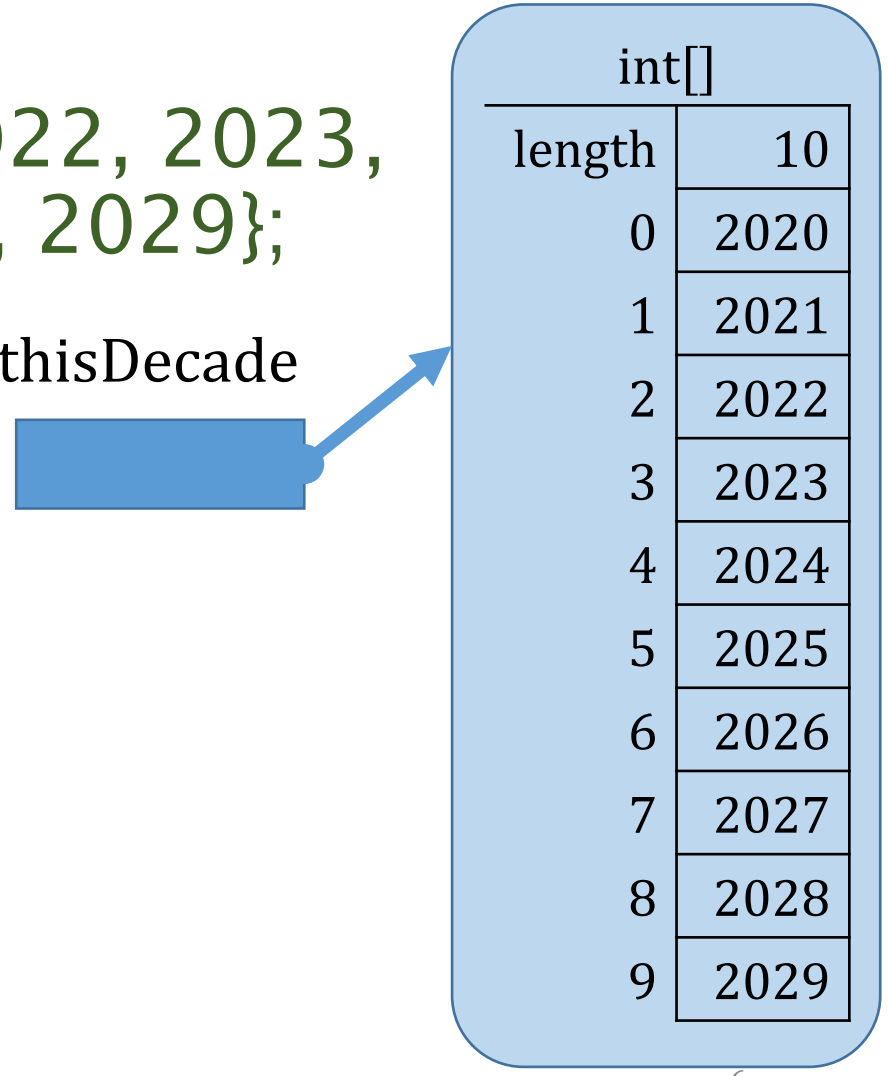
Values of the array

Reference to int[] may refer to "objects" with *different* numbers of fields!

- When created, all values are initialized to zero
- Values can be changed at any time
  - thisDecade[3]=2013;

Shortcut: Declare, Instantiate, & Initialize

```
int[ ] thisDecade = {2020, 2021, 2022, 2023,
    2024, 2025, 2026, 2027, 2028, 2029};
```



Chap. 7.1

Generic Types

- An array is an array
  - array of integers
  - array of Strings
  - array of doubles
  - array of references to objects of the BankAccount class
- Java allows us to create arrays **of** a specific type
  - Type must be specified when declared and instantiated
  - We consider the type part of the “class name” of the array
  - All elements of the array must be of the type specified
  - Arrays don't need them, but Generic Types will eventually be in angle brackets <T>

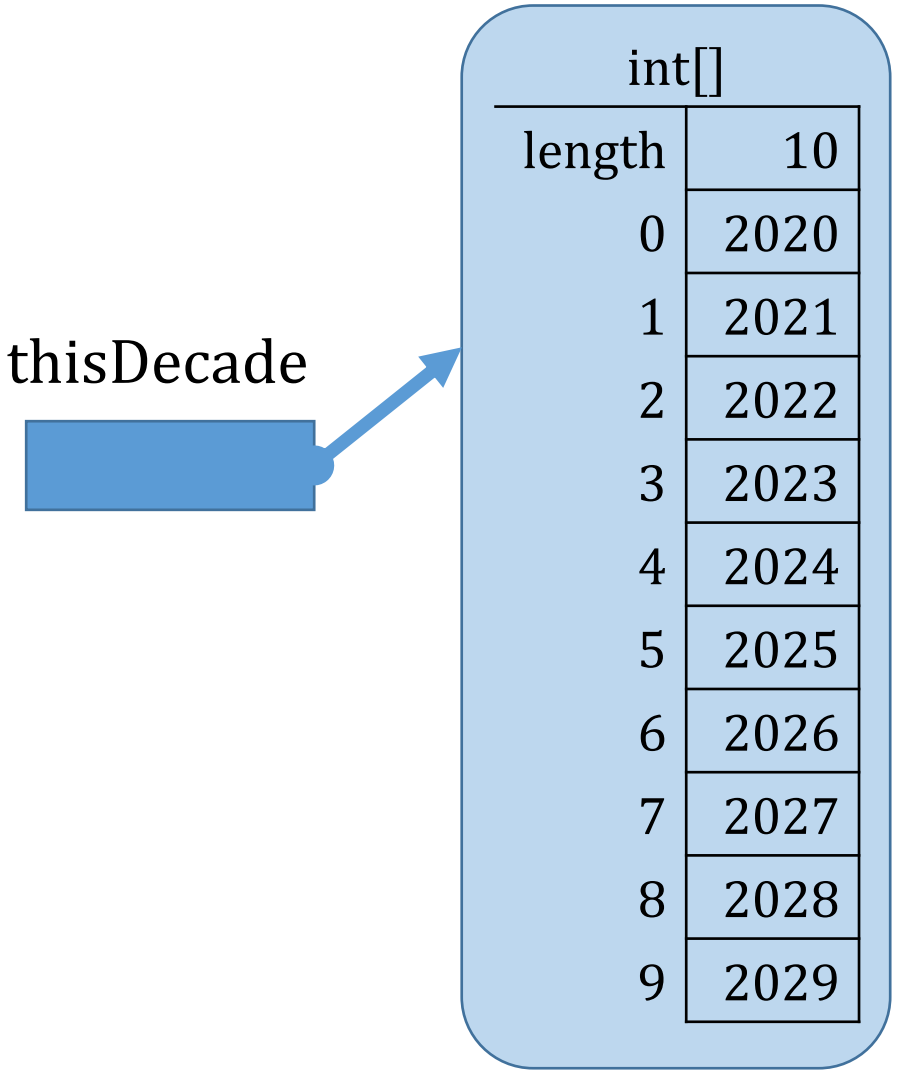
Array Syntax

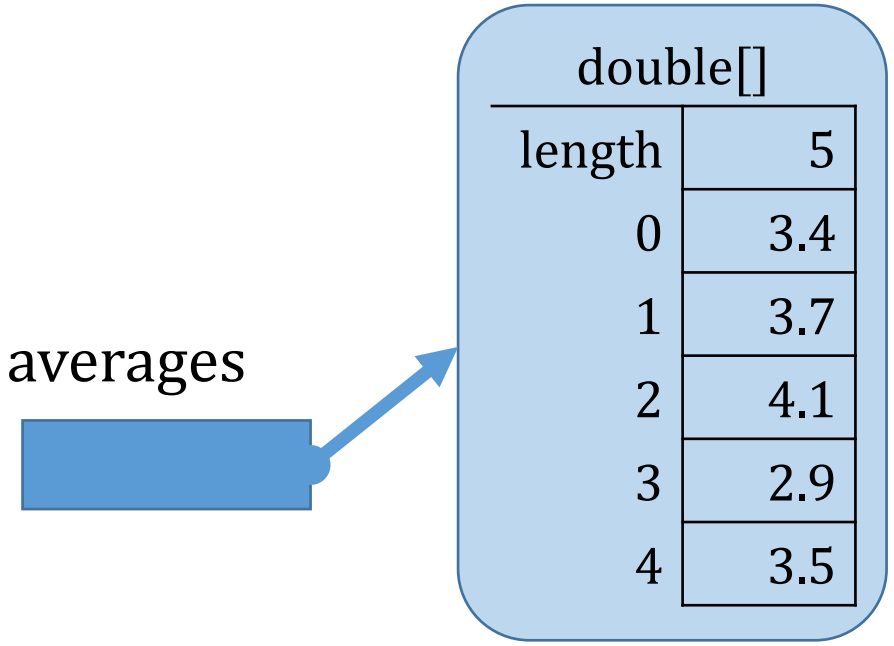
- Declaration: *type*[] *name*;
  - *type* : any built-in type or class name (so far)
- Instantiation: *name* = new *type*[*size*];
  - *size*: integer expression
- Combined: *type*[] *name* = new *type*[*size*];
- Shortcut: *type*[] *name* = { *t1*, *t2*, … };
  - *t1, t2, …* are expressions of the correct type
- Access an element: *name*[*index*]
  - *index*: integer expression, 0 <= *index* < *size*
- Array size: *name*.length

only works in declaration initialization, {} is not a literal array!

name.size() does NOT work

Arrays Objects in Memory





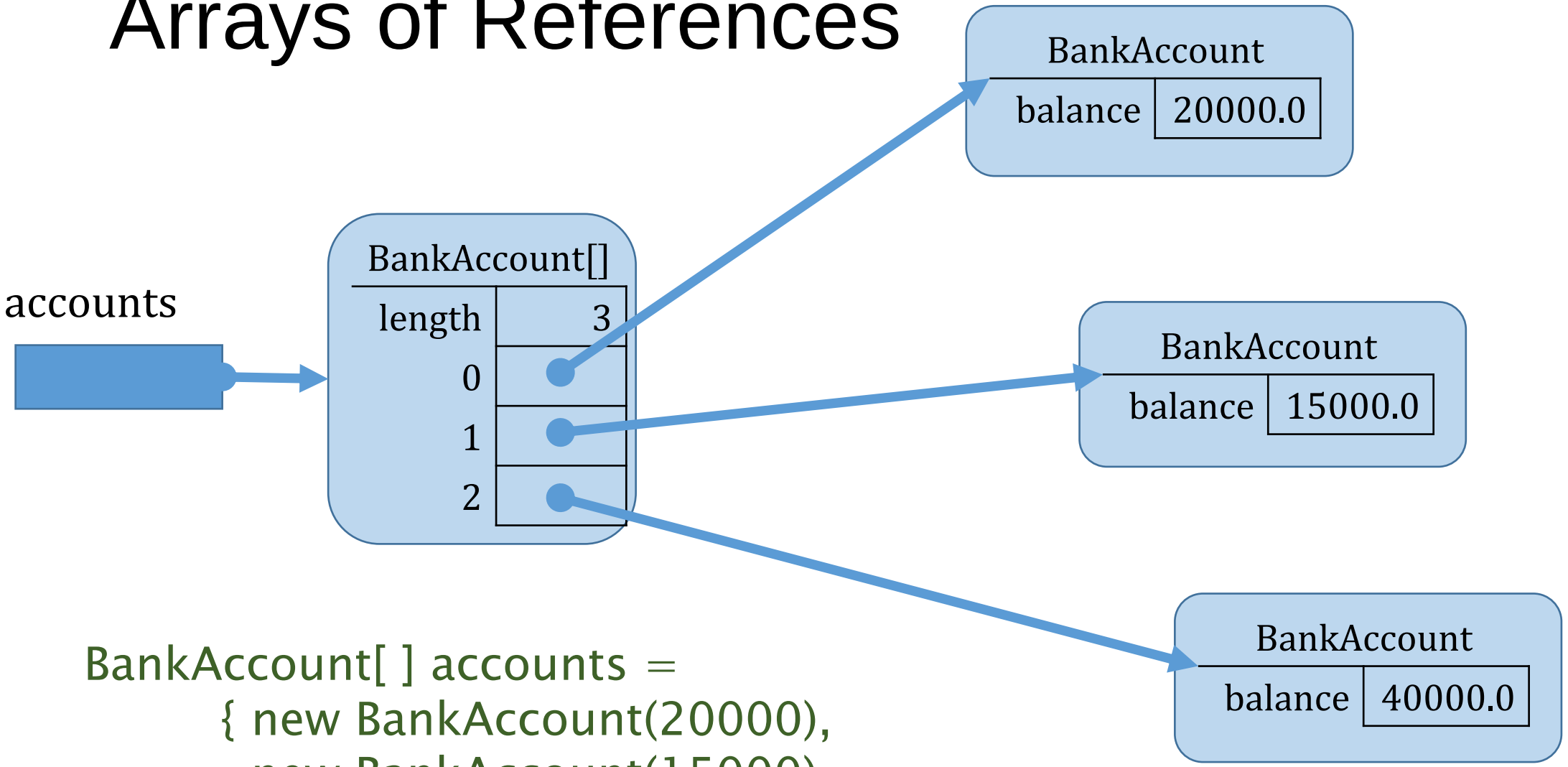

```
double[ ] averages = {3.4, 3.7, 4.1, 2.9, 3.95};
```

Variable names are *not* stored in memory

Arrays of References : Class

```
public class BankAccount {
        private double balance;
        public BankAccount(double firstDeposit) {
                balance = firstDeposit;
        }

        public double getBalance( ) {
                return balance;
        }
        ... code for deposit method
        ... code for withdraw method
}
```

Arrays of References




```
BankAccount[ ] accounts =
        { new BankAccount(20000),
          new BankAccount(15000),
          new BankAccount(40000) };
```

Accessing Elements of Arrays

thisDecade[0] is 2020

averages[averages.length - 1] is 3.95

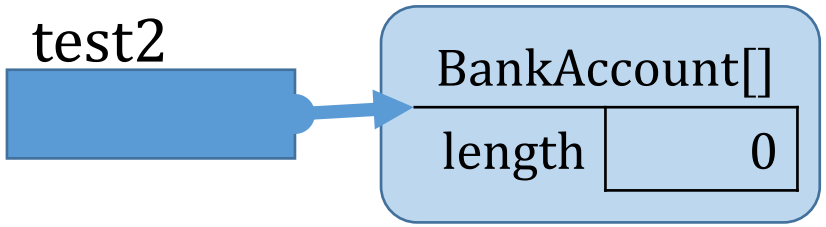
investments[2].getBalance() returns 40000.0

Examples of Arrays of BankAccounts

```
BankAccount[ ] test1 = null;          // uninstantiated array
```

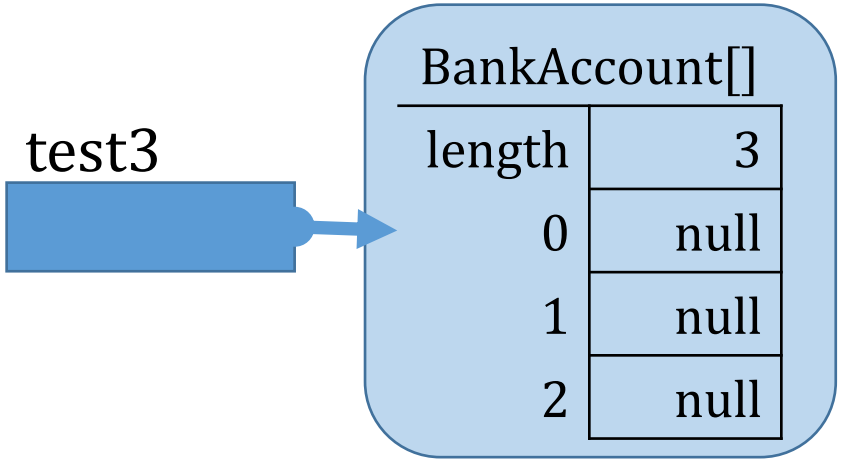
Examples of Arrays of BankAccounts

```
BankAccount[ ] test1 = null;                    // uninstantiated array

BankAccount[ ] test2 = { };                     // empty array
```



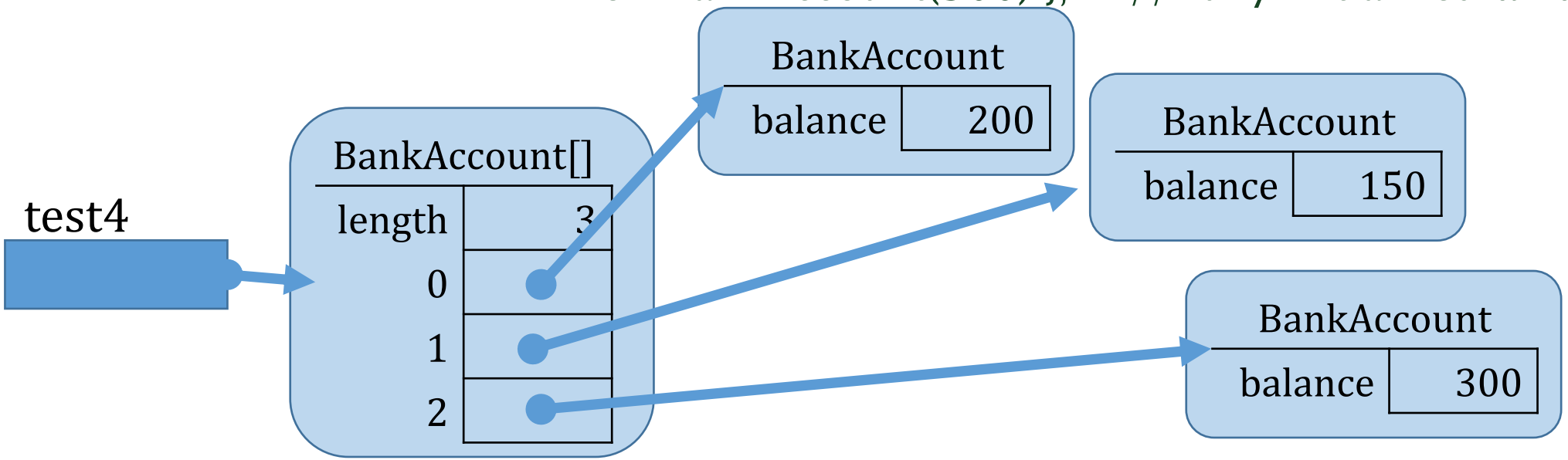
Examples of Arrays of BankAccounts

```
BankAccount[ ] test1 = null;                    // uninstantiated array

BankAccount[ ] test2 = { };                     // empty array

BankAccount[ ] test3 = {null, null, null};      // empty elements
```



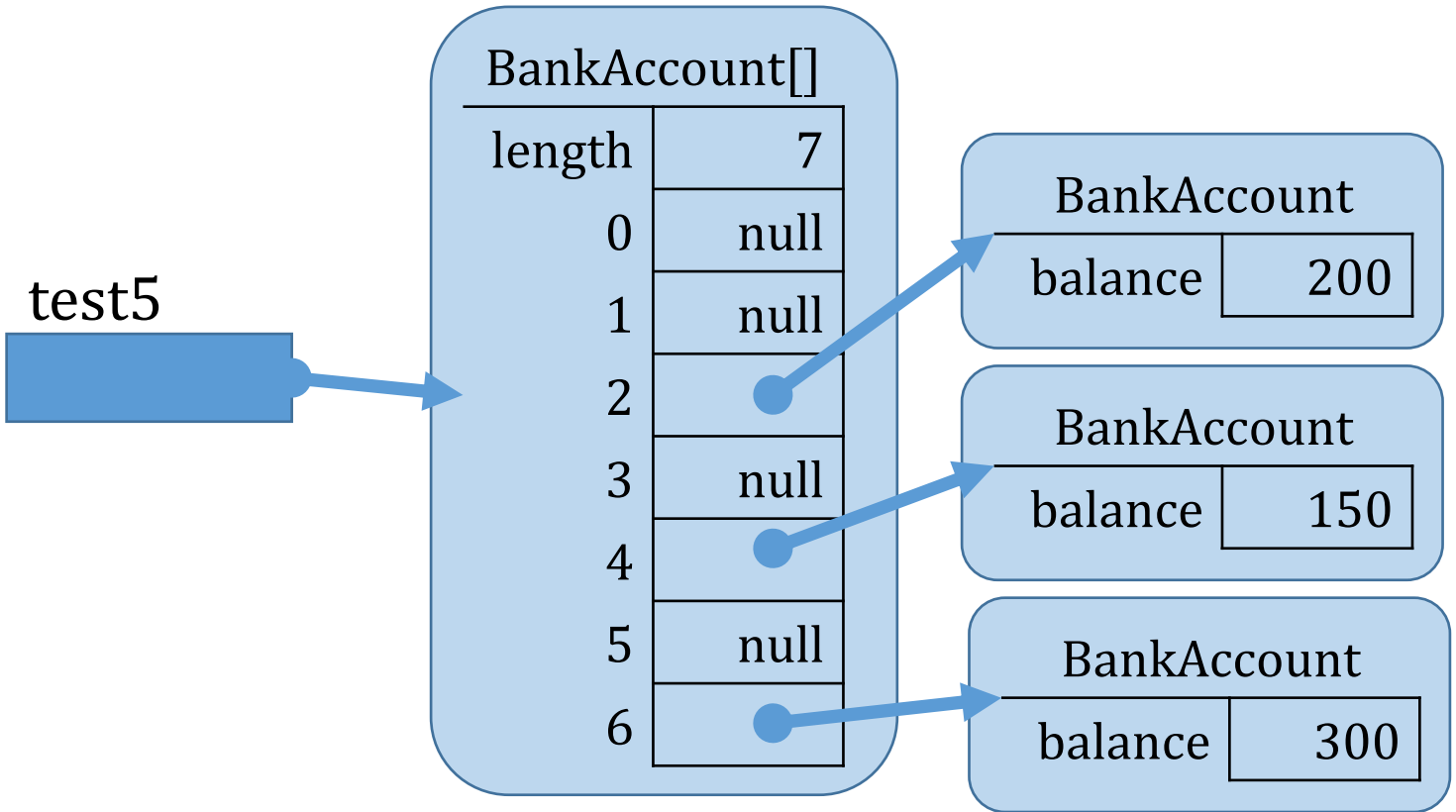
Examples of Arrays of BankAccounts

```
BankAccount[ ] test1 = null;                          // uninstantiated array
BankAccount[ ] test2 = { };                           // empty array
BankAccount[ ] test3 = {null, null, null};            // empty elements
BankAccount[ ] test4 = {new BankAccount(200),
                        new BankAccount(150),
                        new BankAccount(300) };       // fully initialized array
```

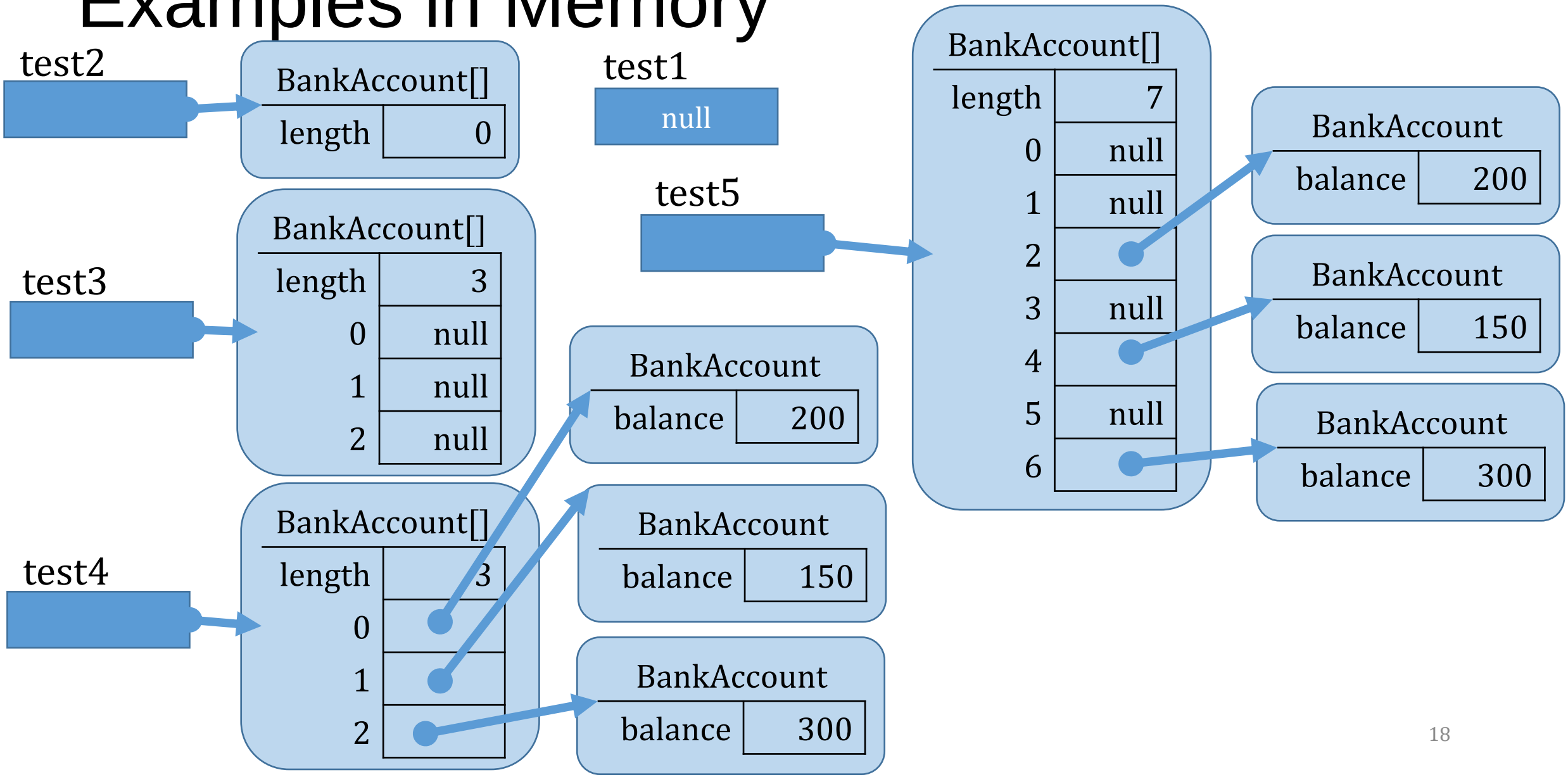


Examples of Arrays of BankAccounts

```
BankAccount[ ] test4 = {new BankAccount(200),new BankAccount(150), new BankAccount(300) };
BankAccount[ ] test5 = {null, null, new BankAccount(200), null,
                        new BankAccount(150), null, new BankAccount(300) };
```



Examples in Memory



Problem with Arrays

- Arrays are great if you know how big they need to be
  - but we don't always know how big it needs to be
- One alternative
  - Start out with medium sized array
  - If it needs to grow bigger, create a bigger array, copy the medium to the bigger array – repeat as necessary
  - If it grows smaller, create a smaller array, copy the medium to the smaller array – repeat as necessary
- Another alternative: Java "ArrayList" class… more to come

Arrays of Arrays

- It is possible to make an array of arrays
  - Not quite the same as multi-dimensional arrays, but close (superset)

- Example:

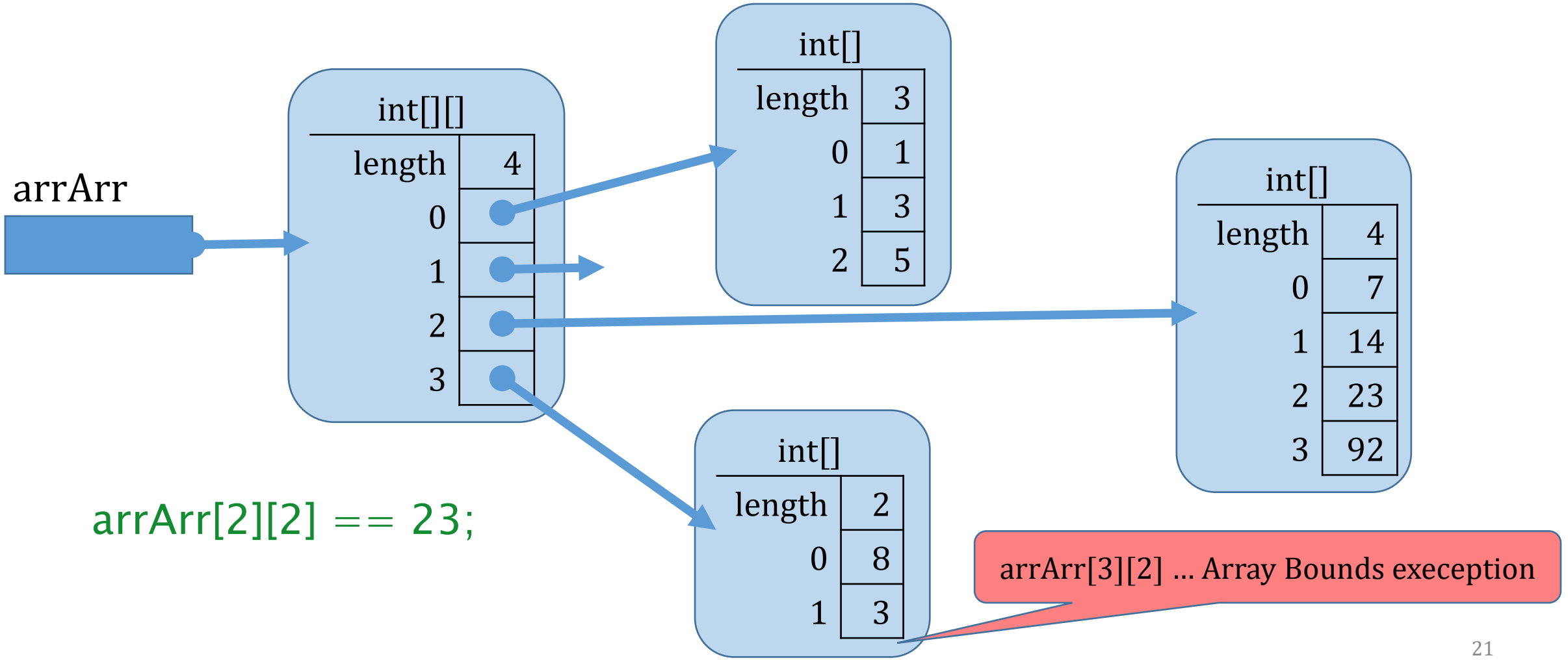
Other languages, like C, have multi-dimensional arrays like a matrix… these are different!

```
int[][] arrArr = { {1, 3, 5} , null, {7, 14, 23, 92}, {8, 3} };
System.out.println("arrArr length is:" + arrArr.length);
```

prints: arrArr length is 4

Memory for Array of Arrays

```
int[][] arrArr = { {1, 3, 5} , null, {7, 14, 23, 92}, {8, 3} };
```




```
arrArr[2][2] == 23;
```

arrArr[3][2] ... Array Bounds exeception

"Arrays" Library Class

- Library class with many static functions to perform on arrays
  - compare, copy, search, sort, fill, select, iterate, stream, toString

- The toString method makes:
  "name[e_0.toString(),e_1.toString(),...e_{s-1}.toString()]"

  invoke as: Arrays.toString(thisDecade)

  returns: thisDecade[2020,2021,2022,2023,2024,2025,2026,2027,2028,2029]