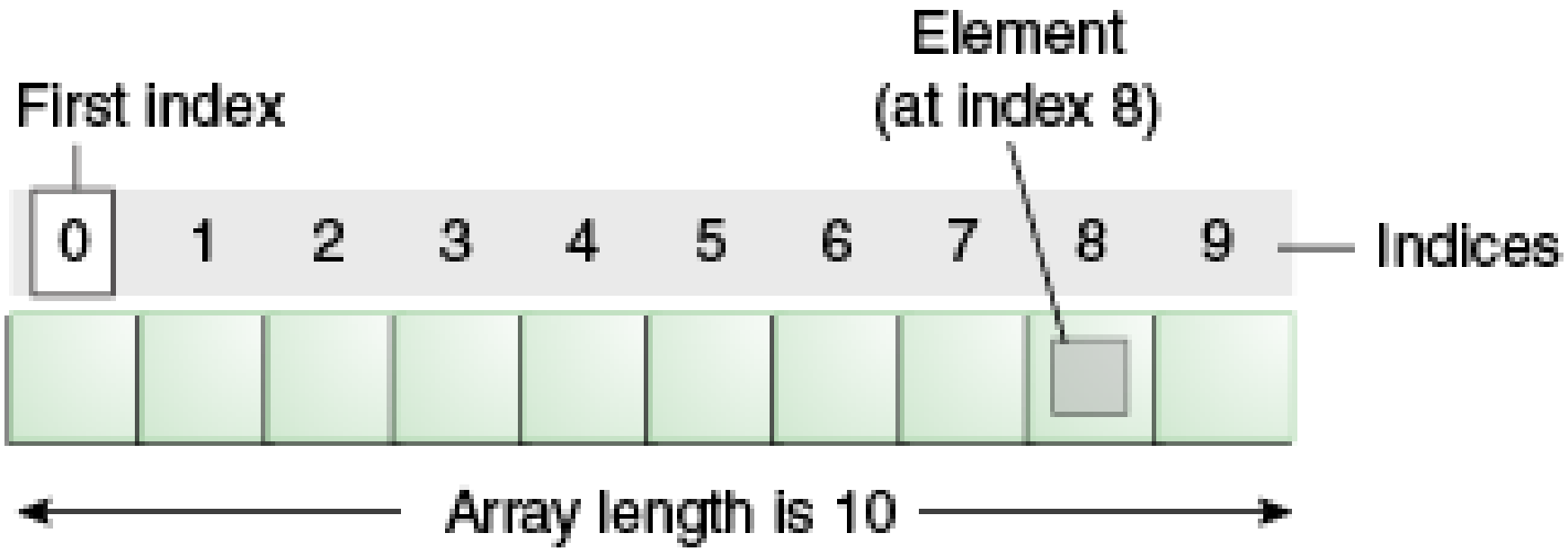
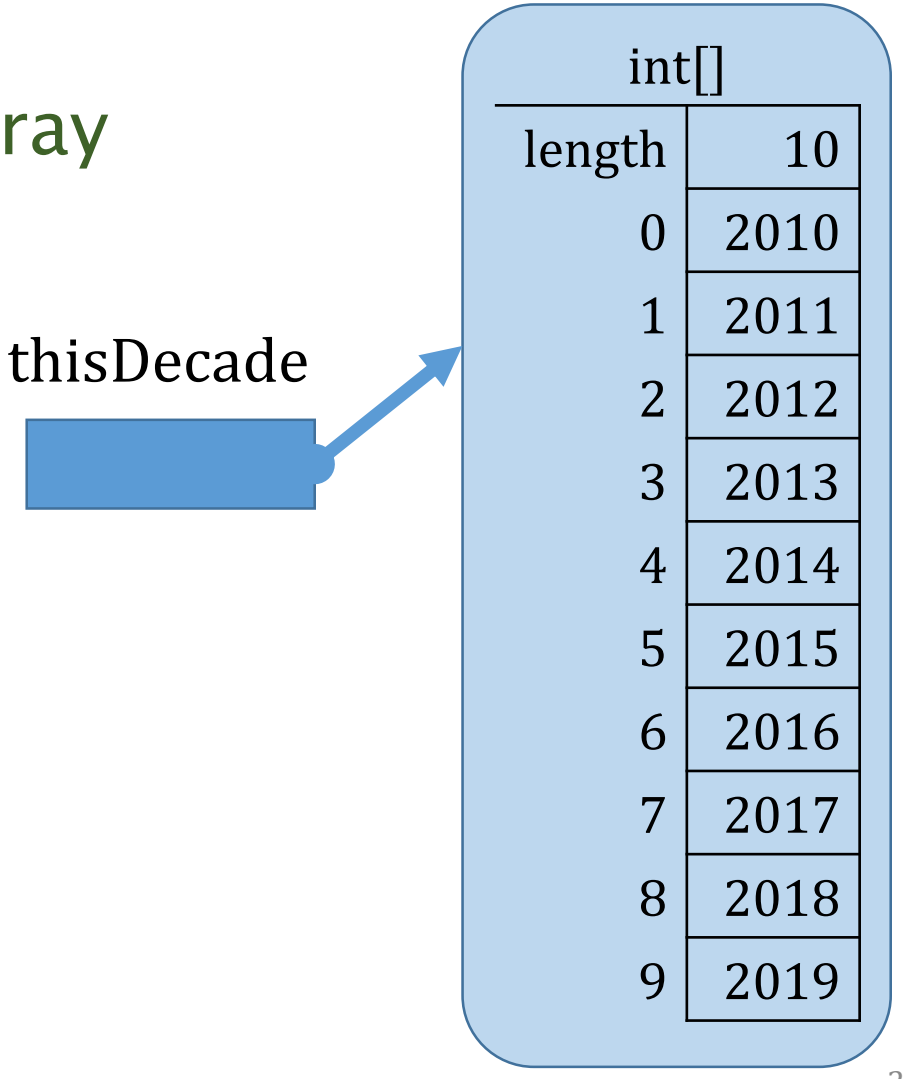
Chap. 7

Arrays



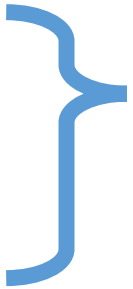
Arrays are Objects

```
int[ ] thisDecade; // Reference to an array
thisDecade=new int[10]; // Instantiate
thisDecade[0]=2010; // Initialize
thisDecade[1]=2011;
…
```



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

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

Shortcut: Declare, Instantiate, & Initialize

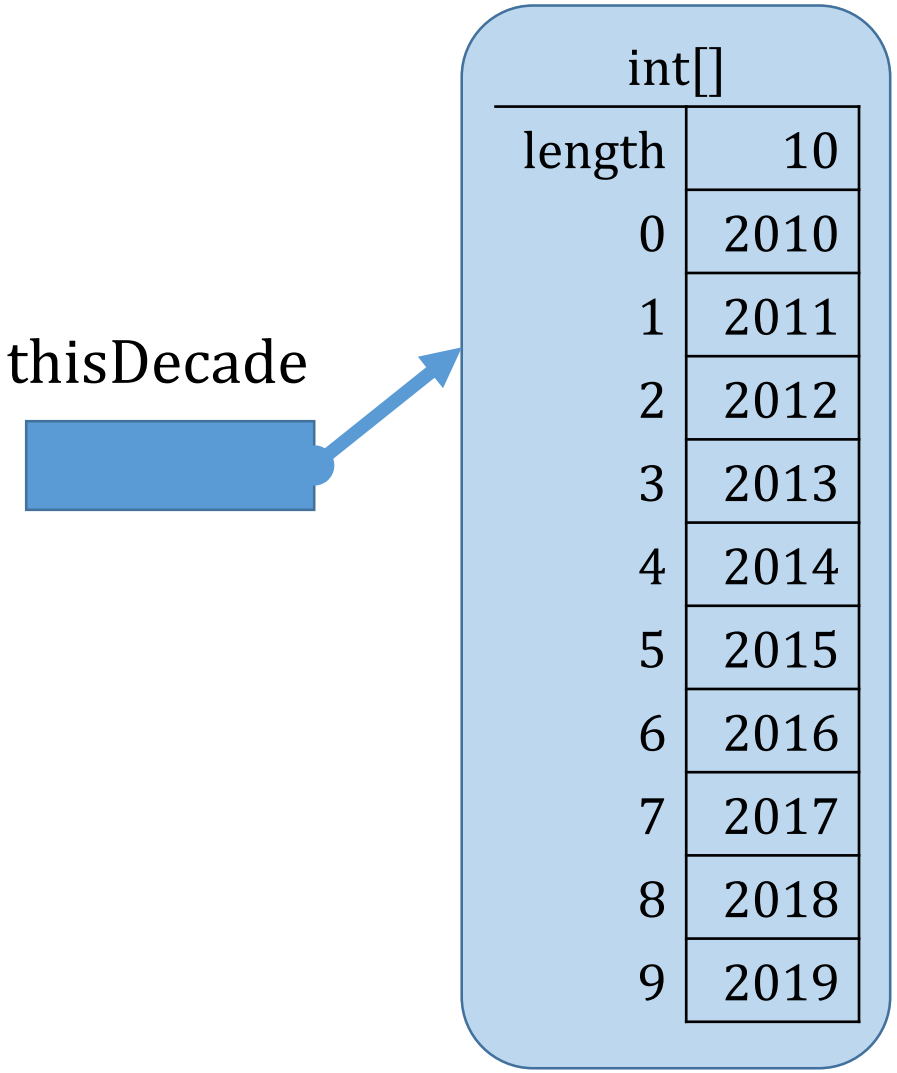
Chap. 7.1

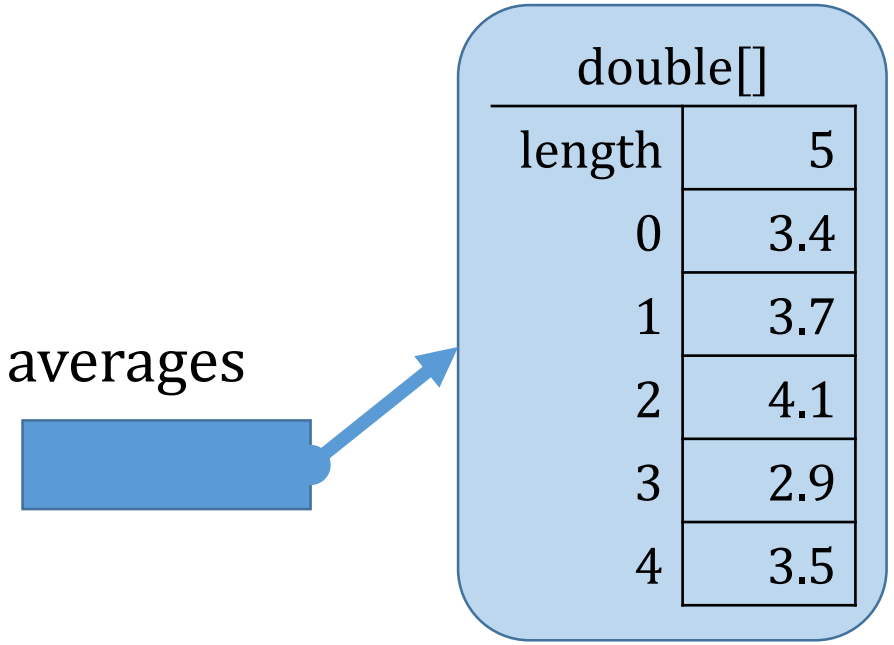
```
String[ ] monthNames = {"January", "February",
      "March", "April", "May", "June", "July",
      "August", "September", "October", "November",
      "December"};
  int[ ] thisDecade = {2010, 2011, 2012, 2013,
      2014, 2015, 2016, 2017, 2018, 2019};
  double[ ] averages = {3.4, 3.7, 4.1, 2.9, 3.95};
  Investment[ ] investments =
      { new Investment(20000, 5.0),
        new Investment(15000, 6.0),
        new Investment(40000, 4.5) };
```

Generic Types

- An array is an array
  - array of integers
  - array of Strings
  - array of doubles
  - array of references to objects of the Investment 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 values must be of the type specified

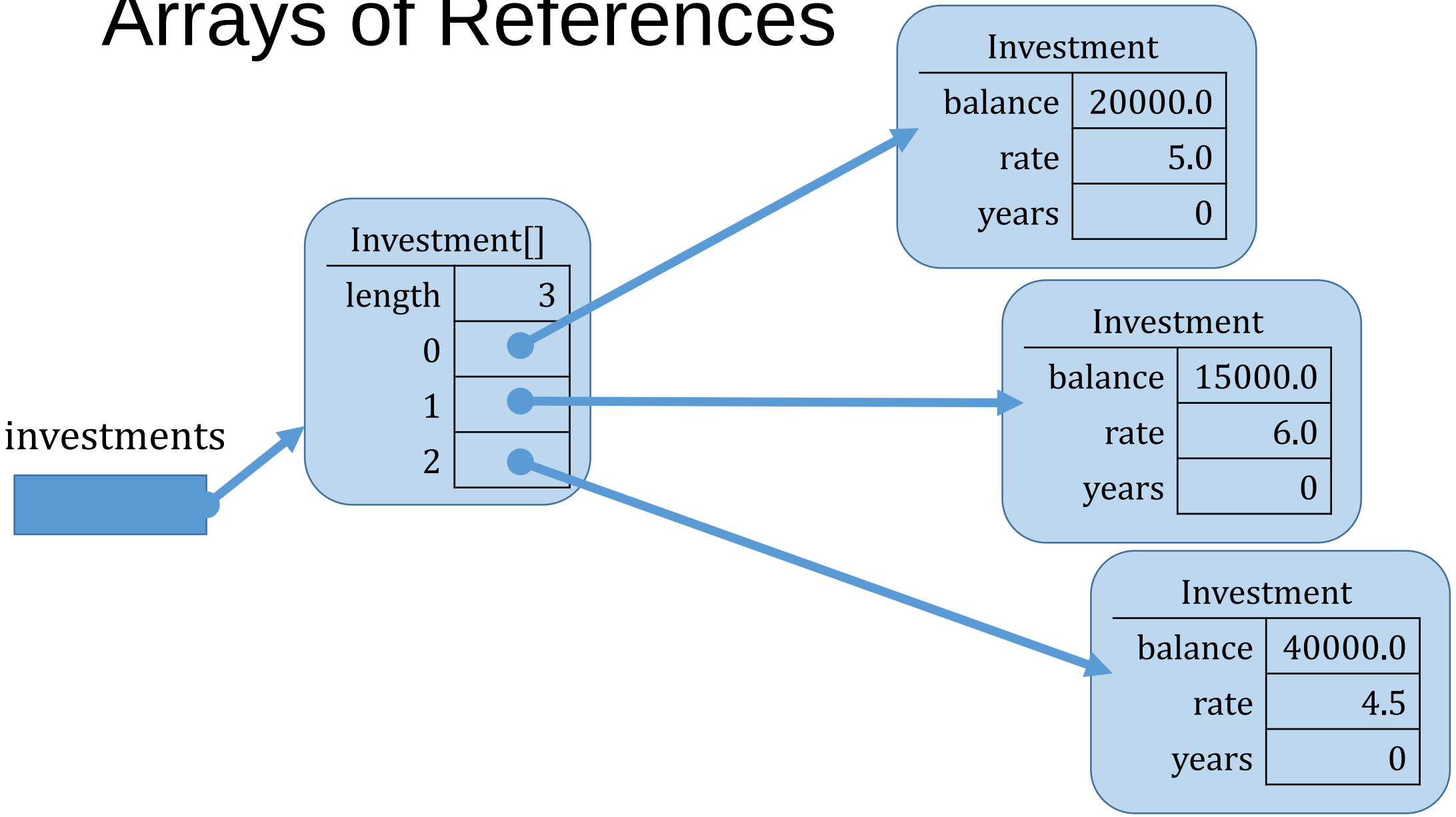
Arrays Objects in Memory





Variable names are *not* stored in memory

Arrays of References



Accessing Elements of Arrays

monthNames[5] is "June“

thisDecade[0] is 2010

averages[averages.length - 1] is 3.95

investments[2].getBalance() returns 15000.0

Array Reference Variables

- arrays are objects that have a field “length” and can only contain that many different values
- An array reference variable can be declared without instantiation
- An array reference variable can be declared and instantiated without explicit initialization
- An array reference variable can be reassigned completely new storage
- An array reference variable can be assigned the value “null”

Declared but not instantiated

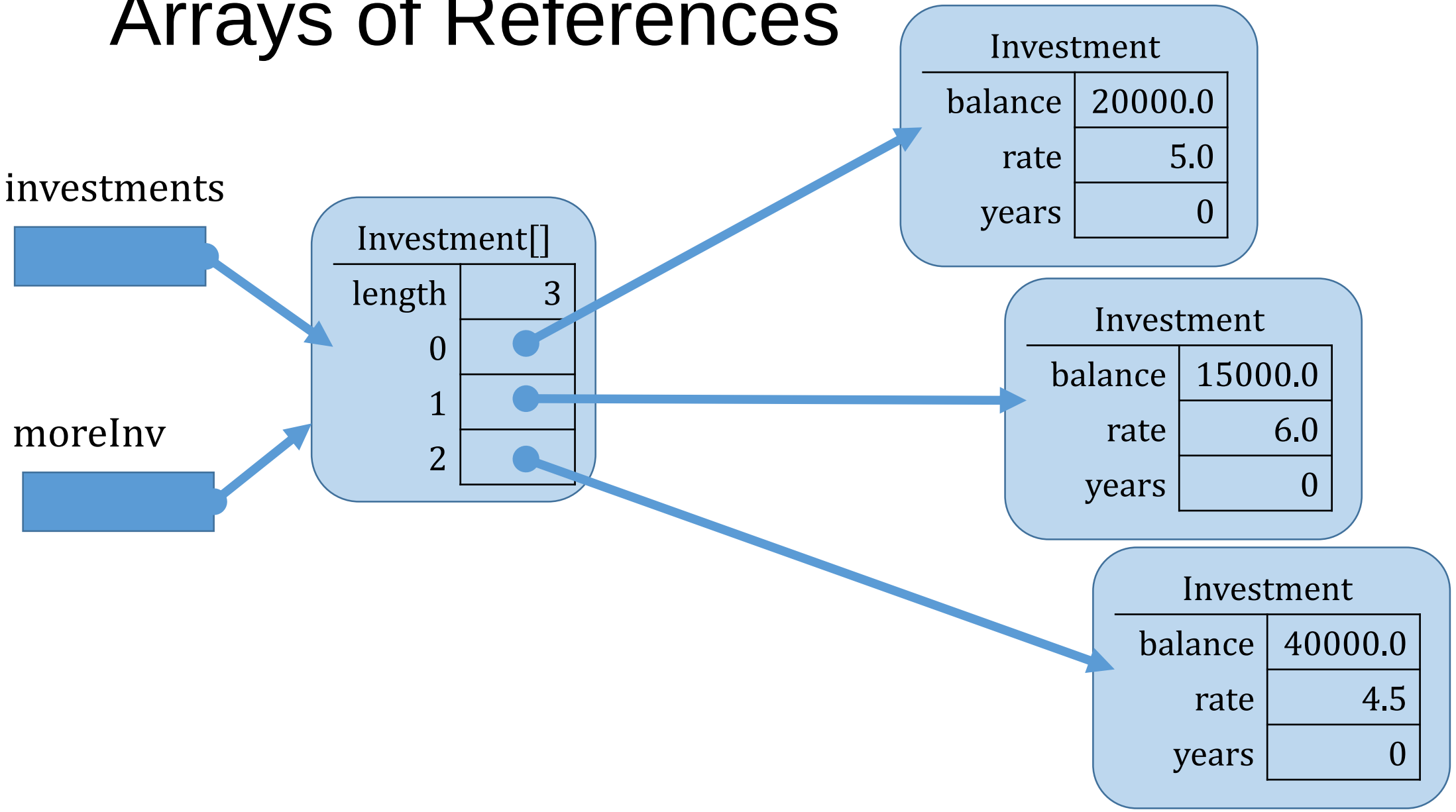
```
Investment[] moreInv;
```

- The value of moreInv is undefined if moreInv is a local variable, or null if moreInv is a field in a class
- We must assign a value to moreInv before we can use it

```
moreInv = investments;
```

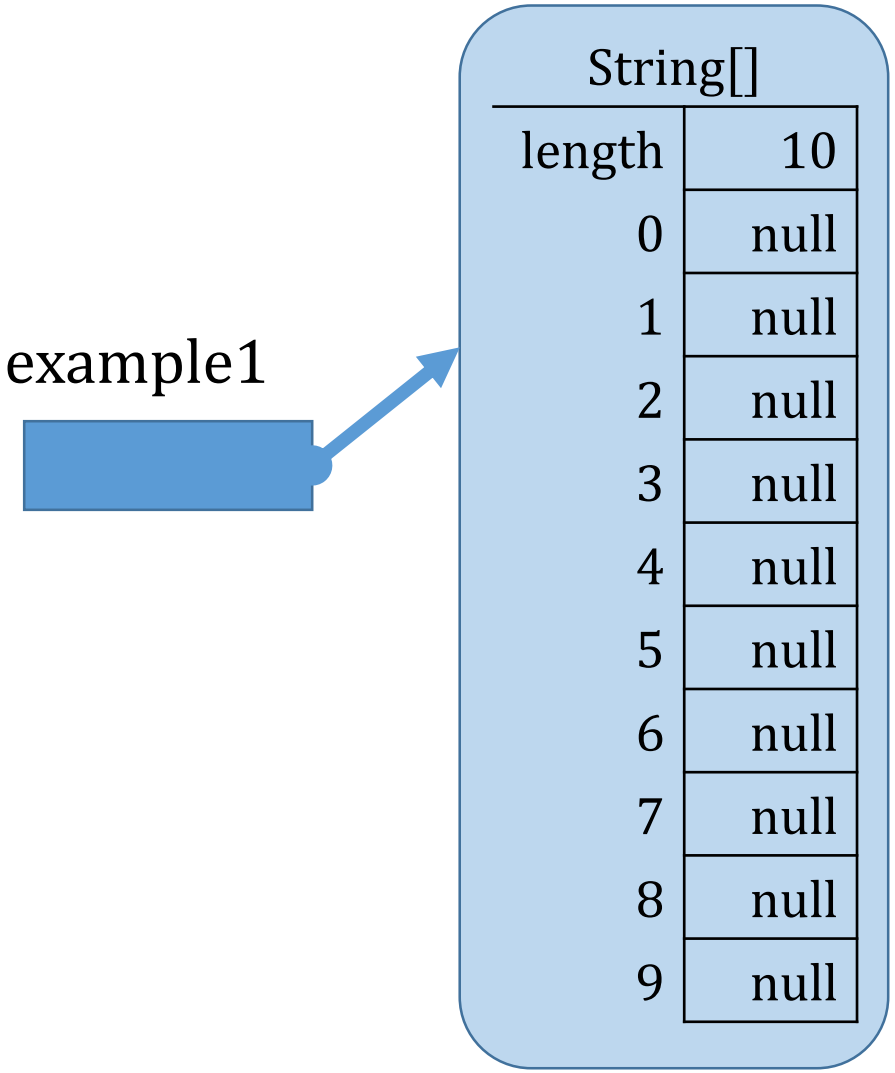
- After assignment, investments and moreInv are two names for the same thing

Arrays of References

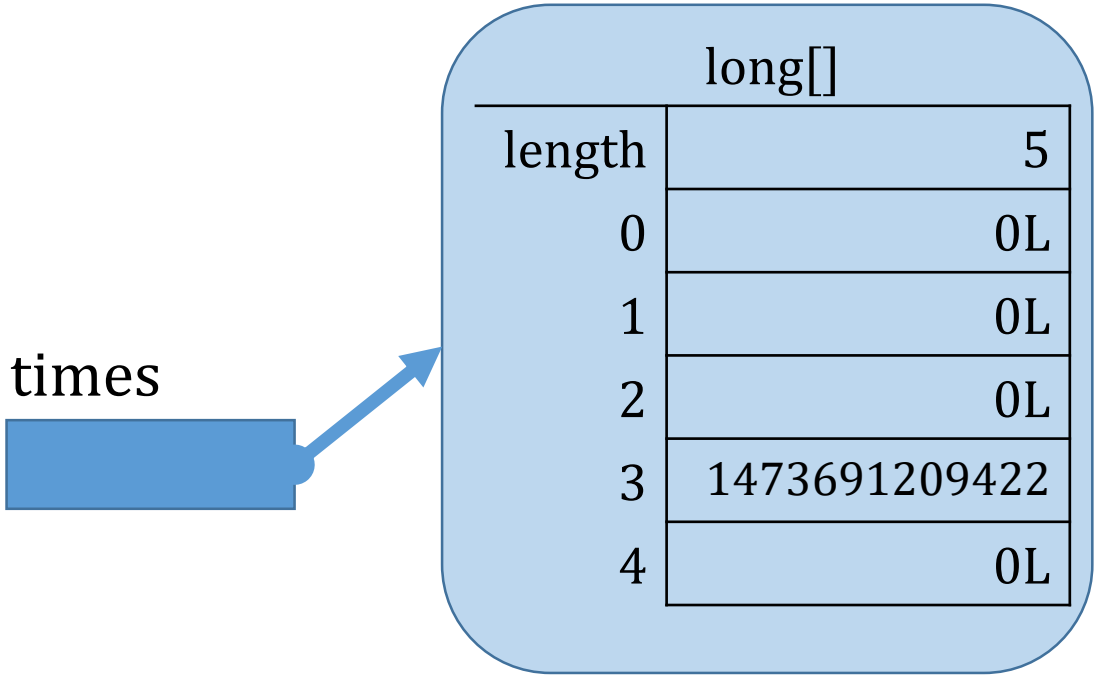


Declared, Instantiated, but not Initialized

```
String[] example1 = new String[10];
```

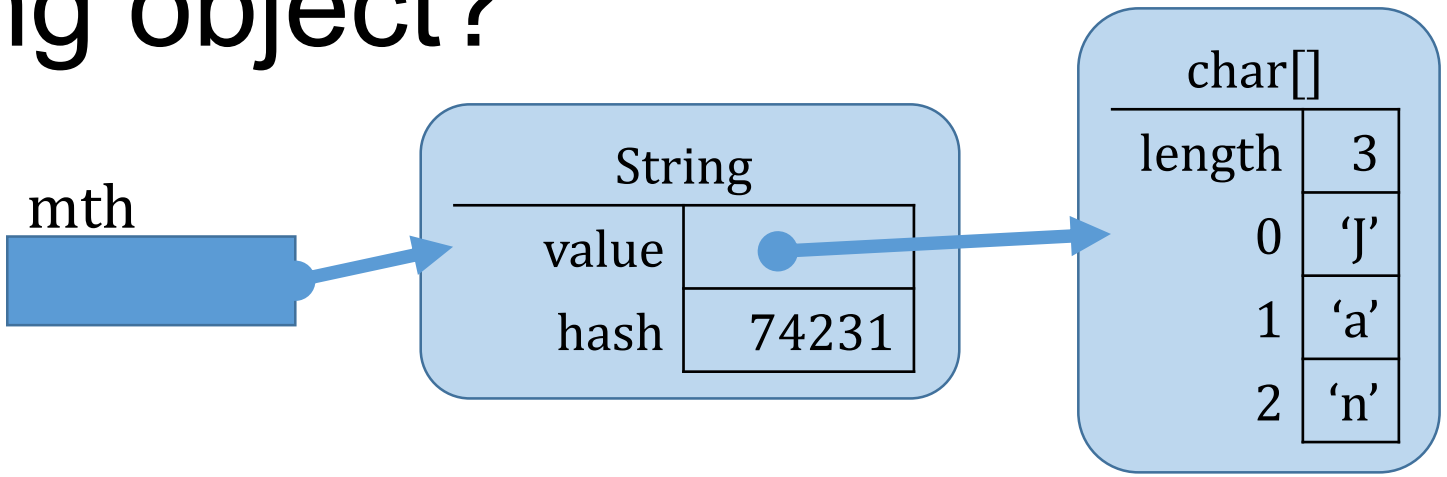



```
long[] times = new long[5];
times[3]=System.currentTimeMillis();
```



What's in a String object?

String mth = "Jan";



- String is an class with two fields…
  - value : Reference to a character array
  - hash : A numeric value that encodes information about the string
    In this example: hash= 31*31*'J' + 31*'a' + 'n' = 74231

    If two strings have different hash values, the strings are different.

Looping Through Arrays

Sect. 6.3

```
public static int sum(int[ ] array) {
        int tot = 0;
        if(array != null) {
                for (int i = 0; i < array.length; ++i) {
                        tot += array[ i ];
                }
        }
        return tot;
}
```

Make sure array is instantiated!

length is a public field in the array class, so we can read it

Enhance "for" loop

Chap. 7.2

```
public static int sum(int[ ] array) {
        int tot = 0;
        if(array != null) {
                for (int t : array) {
                        tot += t;
                }
        }
        return tot;
}
```

like "foreach" in other languages

Enhanced For Loops for Arrays

Chap. 7.2

- Syntactic "sugar"
- If arr is an array of type T: T[] array;

```
for(int i=0;i<array.length; i++) {
        T t = array[i];
        // use t here
}
```

```
for(T t : array ) {
        // t is a copy of array[i]
        // cannot modify array[i]
}
```

Variable Length Parameter Lists

```
public int sum(int... numbers) {
        int tot=0; for(int n : numbers) { tot+=n; }
        return tot;
}
```

- Last item in a parameter list can be of the form *type*... *name*
- When you call, specify ≥ 0 arguments: `int tot = sum(3,4,5);`
- Java creates an array of the specified type: `int[] temp={3,4,5};`
- Java calls the method with a single array argument: `sum(temp);`

Array Minimum

```
public static double minElement(double[ ] array) {
    double minVal = Double.MAX_VALUE;
    if (array != null && array.length > 0) {
        minVal = array[0];
        for (int i = 1; i < array.length; i++) {
            if (array[ i ] < minVal) {
                minVal = array[ i ];
            }
        }
    }
    return minVal;
}
```

Static public variable in Double class

Check for unistantiated or empty

Note: primitive types cannot be null

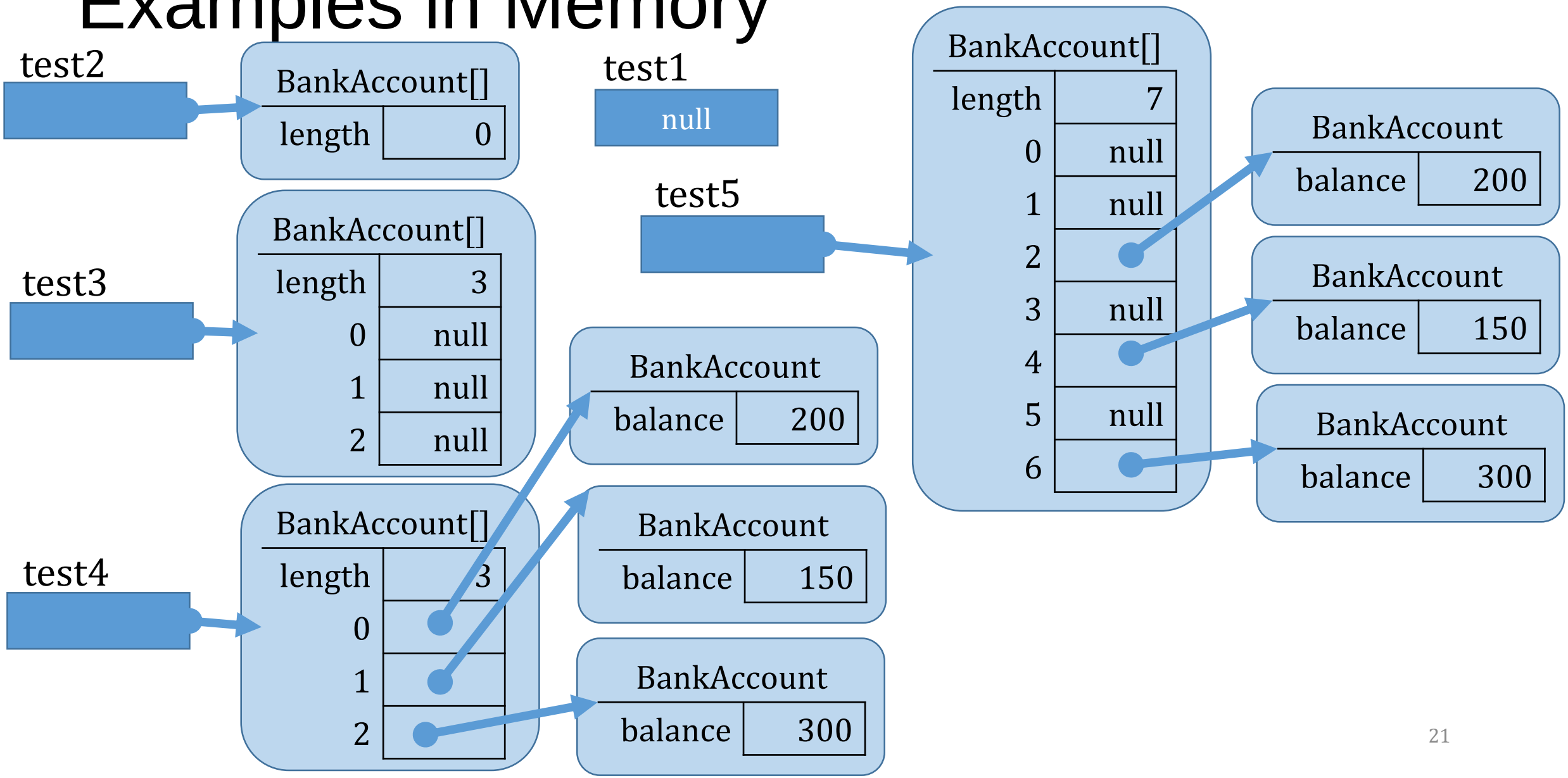
Arrays of References

```
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
}
```

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


BankAccount[ ] test5 = {null, null,
                        new BankAccount(200), null,
                        new BankAccount(150), null,
                        new BankAccount(300)};        // partially initialized array
```

Examples in Memory



Finding the minimum bank account

```
public static BankAccount minBal(BankAccount[ ] arr) {
    BankAccount minAcc = null;
    if (arr != null && arr.length > 0) {
        minAcc = arr[0];
        for (int i = 1; i < arr.length; i++) {
            if (arr[ i ].getBalance( ) < minAcc.getBalance( )) {
                minAcc = arr[ i ];
            }
        }
    }
    return minAcc; // gives null if the array is null or empty
}
```

handle test1 and test2

Null Pointer Exception on test3

Handle null elements

```
public static BankAccount minBal(BankAccount[ ] array) {
   BankAccount minAcc = null;
   if (array != null) {
      for (int i = 0; i < array.length; i++) {
         if (array[ i ] != null) {
            if (minAcc == null){
               minAcc = array[ i ];
            } else if(array[ i ].getBalance( ) < minAcc.getBalance( ))
               minAcc = array[ i ];
            }
         }
      }
   }
   return minAcc; // gives null if the array is null or empty
}
```

Use null to indicate no minimum yet

Don't use array[0] no empty check

Handle null elements extended for loop

```
public static BankAccount minBal(BankAccount[ ] array) {
   BankAccount minAcc = null;
   if (array != null) {
      for (BankAccount acc : array) {
         if (acc != null) {
            if (minAcc == null){
               minAcc = acc;
            } else if(acc.getBalance( ) < minAcc.getBalance( ))
               minAcc = acc;
            }
         }
      }
   }
   return minAcc; // gives null if the array is null or empty
}
```

Find the *index* of the minimum element?

```
public static int indexMin(double[ ] array) {
    int minIndex = -1;
    if (array != null && array.length > 0) {
        minIndex = 0; // 0th element is min so far
        for (int i = 1; i < array.length; i++) {
            if (array[ i ] < array[minIndex]) {
                minIndex = i;
            }
        }
    }
    return minIndex;  // returns -1 if array is null or empty
}
```

Can we use an enhanced for?

If there are multiple mins, returns index of first min.

Change the problem…

- Find the index of the element of an array closest to zero
- For instance:

How can we assign an integer constant to a double array?

```
double[ ] test = {7.1, 3.4, -10.0, 1.5, 5, -4, 7.6, -10, 6.1, -3};
System.out.println(indexMin(test)); // prints 2
System.out.println(indexSmall(test)); // prints 3
```

Find the *index* of the smallest element?

```
import java.lang.Math;
public static int indexSmall(double[ ] array) {
        int minIndex = -1;
        if (array != null && array.length > 0) {
                minIndex = 0;
                for (int i = 1; i < array.length; i++) {
                        if (abs(array[ i ]) < abs(array[minIndex])) {
                                minIndex = i;
                        }
                }
        }
        return minIndex;  // returns -1 if array is null or empty
}
```

abs(n) defined for a variety of numeric types.

Finding out More about abs(n)

- Use the online Javadoc for the system library: http://docs.oracle.com/javase/8/docs/api/index.html
- Look at the source code in src.zip:
  - In Windows, start at C:\ProgramFiles\Java\jdk1.8.0_131
  - If you don't have it, reinstall the compiler but check that you want the source code during the install step
  - On UNIX (if installed) in /usr/bin/jdk1.8.0_*/src.zip
- Use the “import” value to find the file: java/lang/Math.java

Problem with Arrays

- Arrays are great if you know how big they need to be when you write the code, 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
  - Lots of array copying and wasted space
- Another alternative: Java "ArrayList" class… more to come