Java Programming

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

Today's Lecture

- Arrays

Review - Memory

```
public class Date {
        public int year;
        public int month;
        public int day;
}
public class Employee {
        public int id;
        public Date hired = new Date();
}
public static void main(String[] args)
{
        int a;                      // Which memory area?
        Employee e;                 // Which memory area?
        e = new Employee();         // Which memory area?
}
```

Questions

How much memory is allocated on the **HEAP** assuming 4 byte integers and 4 byte pointers?

On the **STACK**?

Can a pointer be on the **HEAP**?

Review - Memory

```
public class Date {
        public int year;
        public int month;
        public int day;
}
public class Employee {
        public int id;
        public Date hired = new Date();
}
public static void main(String[] args)
{
        int a;                      // Which memory area?
        Employee e;                 // Which memory area?
        e = new Employee();         // Which memory area?
}
```

Questions

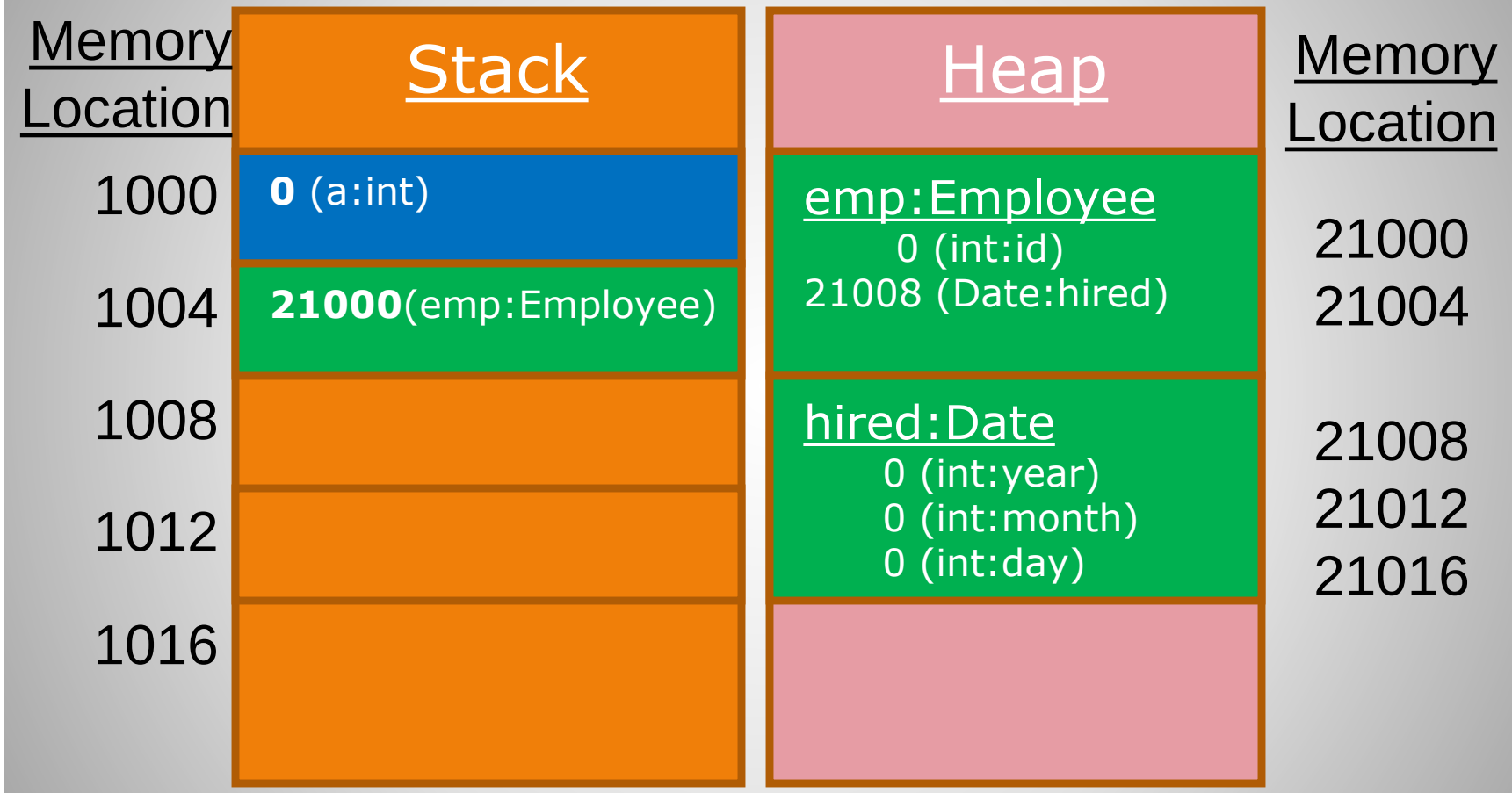
STACK = 8 bytes

HEAP = 20 bytes

Can a pointer be on the **HEAP**?

YES

- ***new is called for Employee.***



Review - Memory

Data Structures

- What is a data structure?
- Collections of related data items.
- Arrays are data structures consisting of related data items of the ***same*** type.
- For example: all ints, all Strings, all Person objects etc.

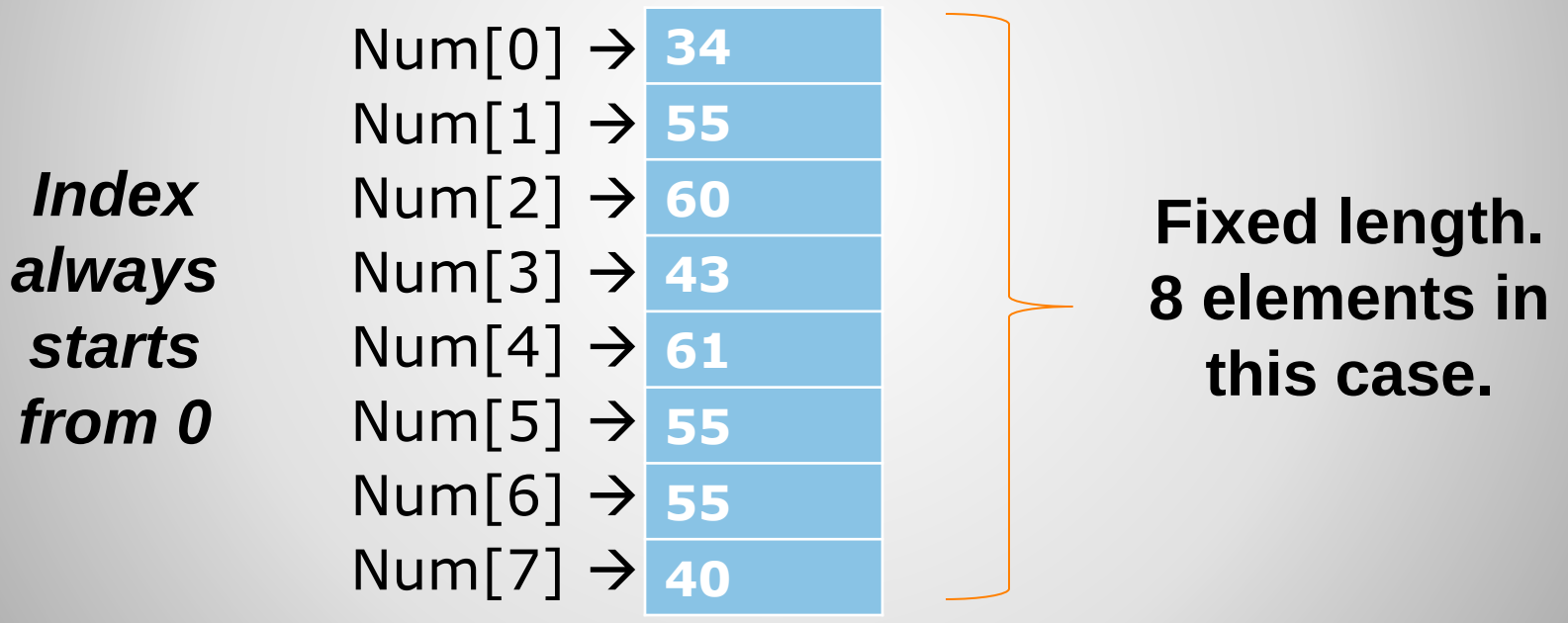
Arrays

- Arrays have a fixed length (cannot change the size).
- Once an array is created you cannot change its size (there is a way around this though).
- An array variable may be ***reassigned*** such that it refers to a ***new*** array of a ***different*** size.

Arrays

- Array entries are called ***elements*** (or components).
- Elements of an array can be either primitive or reference types.
- The array itself is considered a reference type because it is an object.

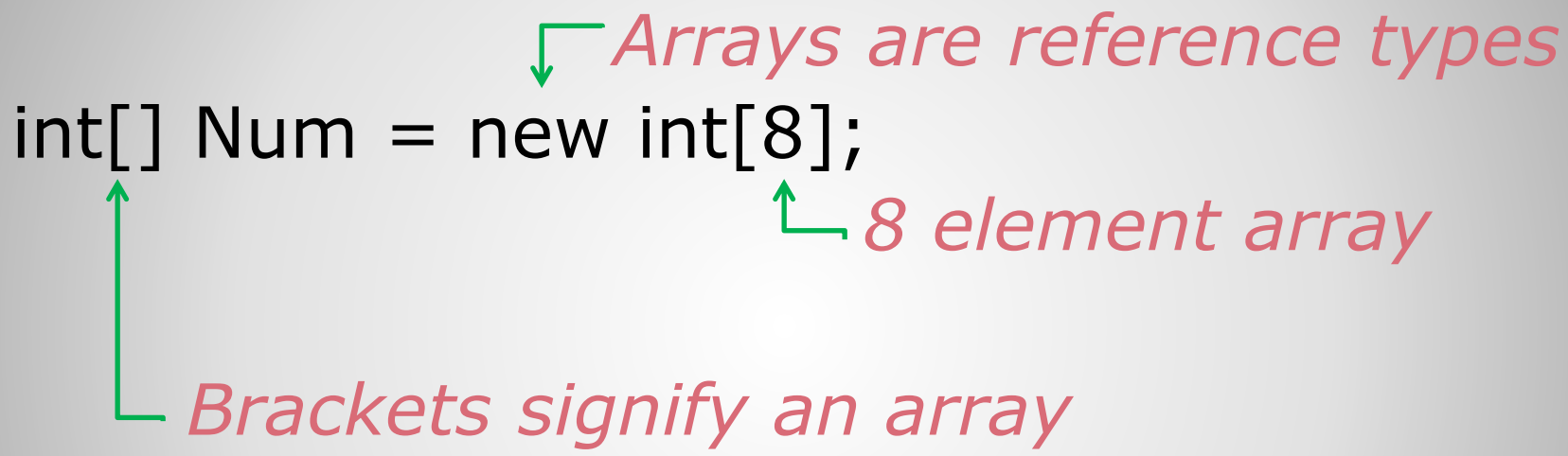
- Example Array:
- This array is called ***Num***. It has 8 elements.



Arrays

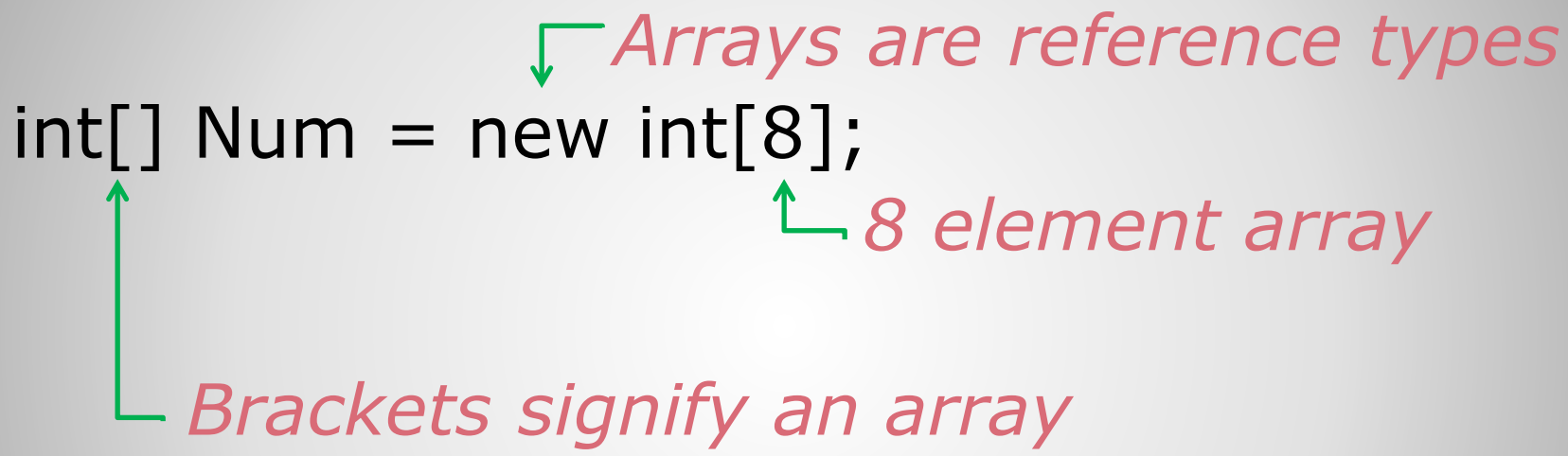
Arrays

- Declare an 8 element integer array called Num:



Arrays

- Declare an 8 element integer array called Num:



- Could also declare and allocate memory in ***two steps***:

```
int[] Num;  // Num variable

Num = new int[8]; // Allocate heap memory

Num = new int[16]; // Allocate heap memory
```

Arrays

Arrays

- ***Setting*** values in an array.
- Java arrays are "offset 0".
- This means that you ***start numbering from 0 instead of 1***.

- ***Setting*** values in an array.

- Set the value of the ***first*** element of the array called Num to 66:

```
int[] Num = new int[8];
Num[0] = 66;  // Index 0 is 1st element
```

Arrays

- ***Setting*** values in an array.

- Set the value of the ***third*** element of the array called Num to 75:

```
int[] Num = new int[8];
Num[2] = 75;  // Index 2 is 3rd element
```

Arrays

Arrays

- ***Getting*** values in an array.

- Get the value of the ***first*** element in the array:

```
int[] Num = new int[8];
int score;

score = Num[0]; // Get value of 1st element.
                // Index of 1st element is 0.
```

- ***Getting*** values in an array.

- Get the value of the ***third*** element in the array:

```
int[] Num = new int[8];
int score;

score = Num[2]; // Get value of 3rd element.
                // Index of 3rd element is 2.
```

Arrays

Arrays

- To get the number of elements in an array use the following code:

```
int[] Num = new int[8];

System.out.println(Num.length);
```

- Do in-class problem for ch 7 (problem 1).

In-Class Problem

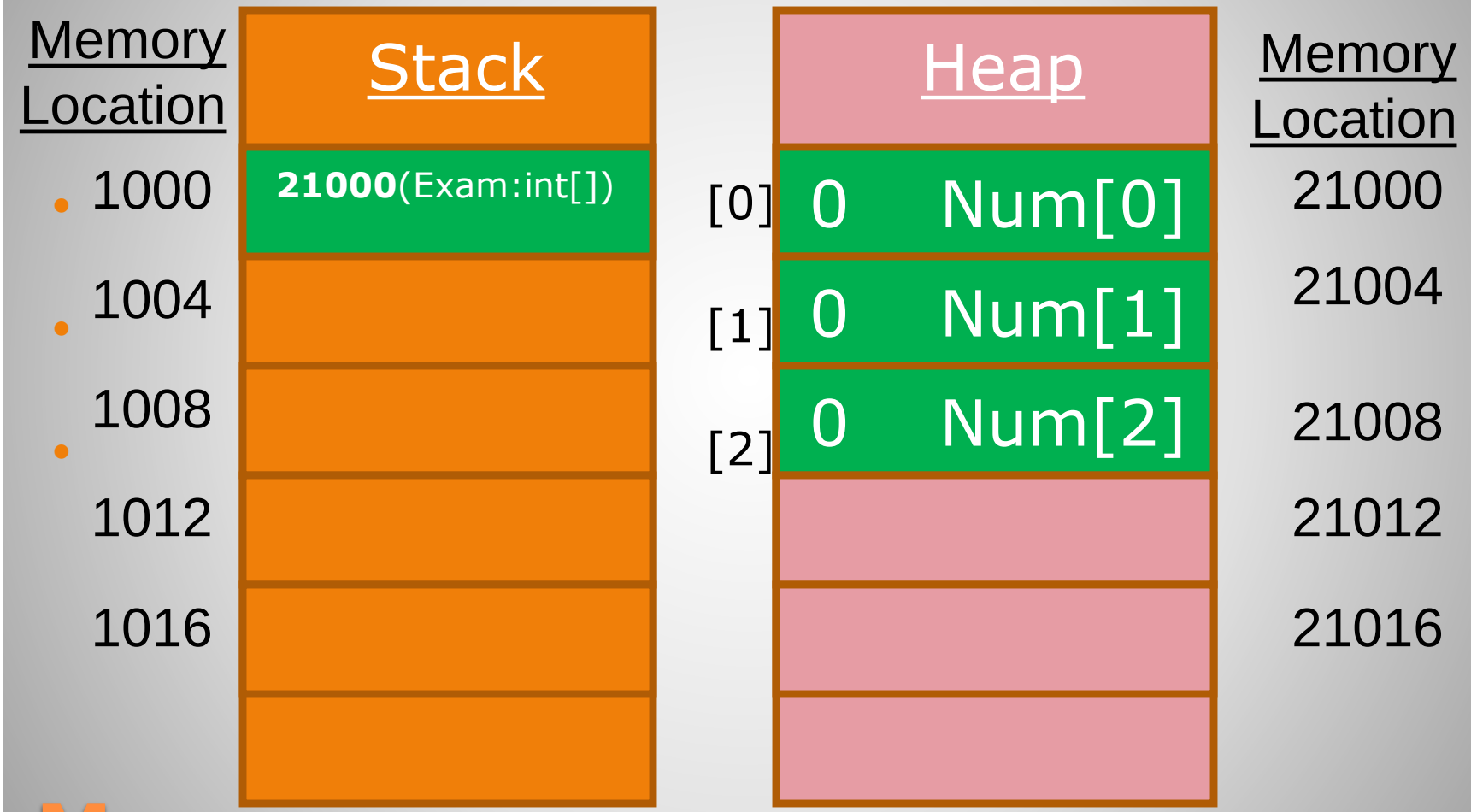
- Do in-class problem for ch 7 (problem 2).

In-Class Problem

Arrays

- Arrays are ***reference*** types.
- You ***MUST*** call ***new*** to initialize them.
- What does an array look like in memory?

- *int[] Exam = new int[3]; // 3 elements in array*



Memory

- Do in-class problem for ch 7 (problem 3).

In-Class Problem

- **Array Initialization**.

```
// Allocate AND initialize a 3 element array
int[] Num = { 75, 82, 95 };
```

Note: new is automatically called for you!

Arrays

- Do in-class problem for ch 7 (problem 4).

In-Class Problem

Arrays

- Arrays of reference types.
- An array itself is a reference type.
- Previously, we created an array of integers. These array elements were all primitive.
- We can also create arrays containing objects that are references types.

- Now let's create a Person type:

```
class Person
{
  private String m_Name;

  public Person(String name) { m_Name = name;}

  public String GetName() { return m_Name; }
  public void SetName(String name)
  { m_Name = name; }
}
```

Arrays

Arrays

- Create an array containing Person objects

```
// Allocate the 3 element array of Person
Person[] group = new Person[3];

// Call new FOR EACH element of the array
group[0] = new Person("Arthur");
group[1] = new Person("Aidan");
group[2] = new Person("Gareth");
```

- Do in-class problem for ch 7 (problem 5).

In-Class Problem

Arrays

- Two-dimensional arrays
- Arrays of Arrays
- An array where ***each element*** is an ***array***.
- How do you declare a two-dimensional array?

Code to create a two-dimensional array called "a":

```
int[][] a;
a = new int[3][4];
```

Here is what the array would look like:

[0][0]	[0][1]	[0][2]	[0][3]
[1][0]	[1][1]	[1][2]	[1][3]
[2][0]	[2][1]	[2][2]	[2][3]

There are ***three rows*** and ***four columns***.

Arrays

Arrays

- Declare and initialize a two-dimensional array:

```
int[][] nums= {      {20, 30, 40},
                     {50, 40, 20},
                     {70, 10, 30}  };
```

- This array has three rows and three columns.
- How do you access elements of a two-dimensional array?

- Need to specify the index for ***both the row and the column.***

  nums[0][2]

  This will access ***row 0*** and ***column 2*** of the array variable ***num***.

- How do you print all the elements of a two-dimensional array?

Arrays

- Print all the elements of the two-dimensional array:

```
for (int i=0; i<nums.length; i++)
{
    for (int k=0; k<nums[i].length; k++)
    {
        System.out.println(nums[i][k]);
    }
}
```

Arrays

Jagged Array

- Each row in a Java two-dimensional array does NOT have to be the same size.
- This is called a jagged array.
- For example…

```
int[][] myA = { { 3, 4, 5 }, { 77, 50 }};

for (int i = 0; i < myA.length; i++) {
        for (int j = 0; j < myA[i].length; j++)
        {
                System.out.print(myA[i][j] + " ");
        }

        System.out.println();
}
```

Jagged Array Example

Employee class definition

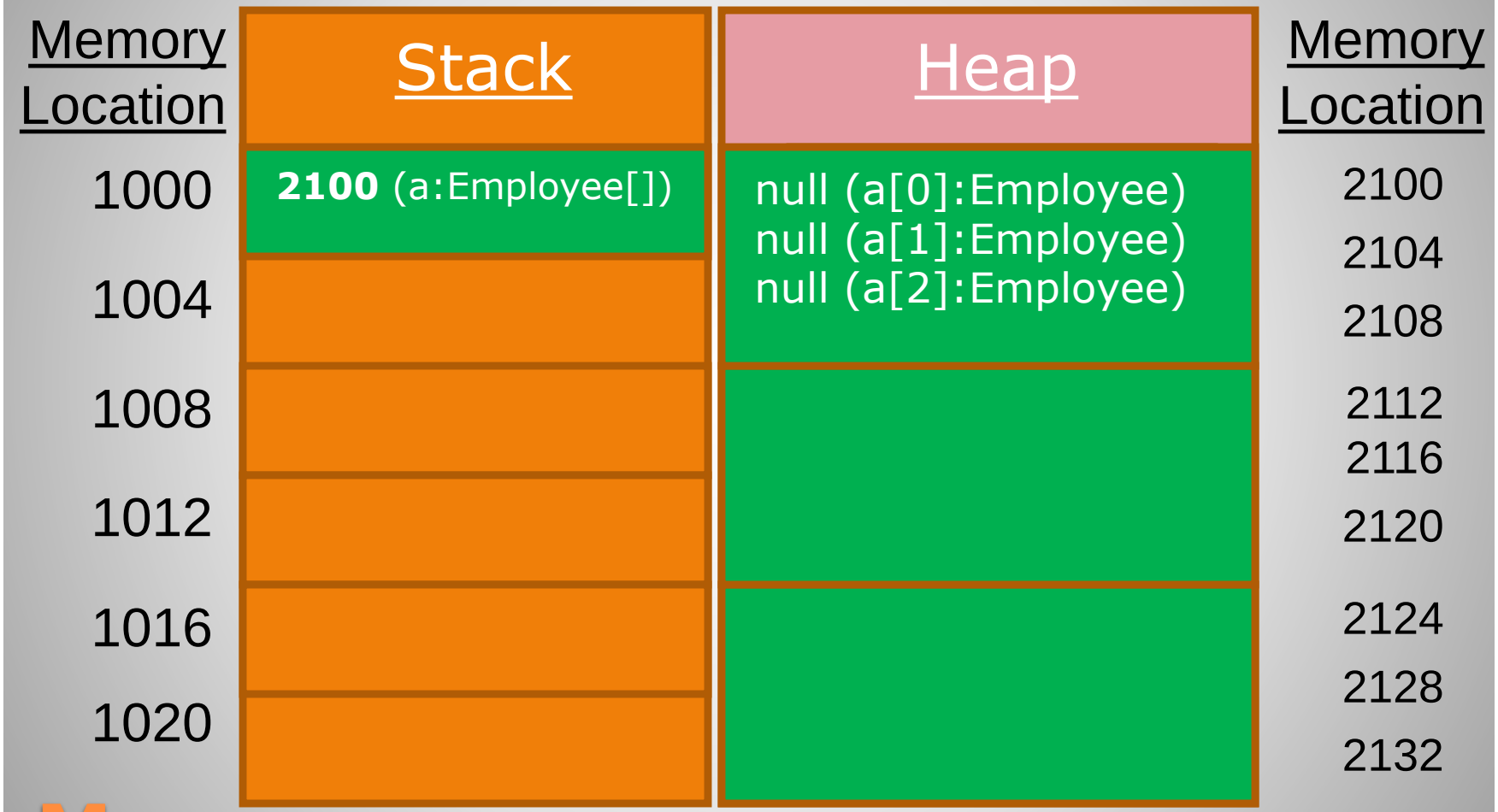
```
class Employee {
    private int m_Id;

    public int GetId () {
            return m_Id;
    }

    public void SetId(int id) {
            m_Id = id
    }
}

// The following code is located in main in another file…
Employee[] a;
a = new Employee[3];

a[0].SetId(100);
```

Call new for array

What will happen when this runs?

Memory

NullPointerException – NEW NOT CALLED ON a[0]



Employee class definition

```
class Employee {
    private int m_Id;
    private int m_Dept;

    public int GetId () { return m_Id; }
    public void SetId(int id) { m_Id = id; }
    public int GetDept () { return m_Dept; }
    public void SetDept(int dept) { m_Dept = dept; }
}
// The following code is located in main in another file…
Employee[] a;
a = new Employee[3];

a[0] = new Employee();
a[1] = new Employee();
a[2] = new Employee();

a[0].SetId(100);
```

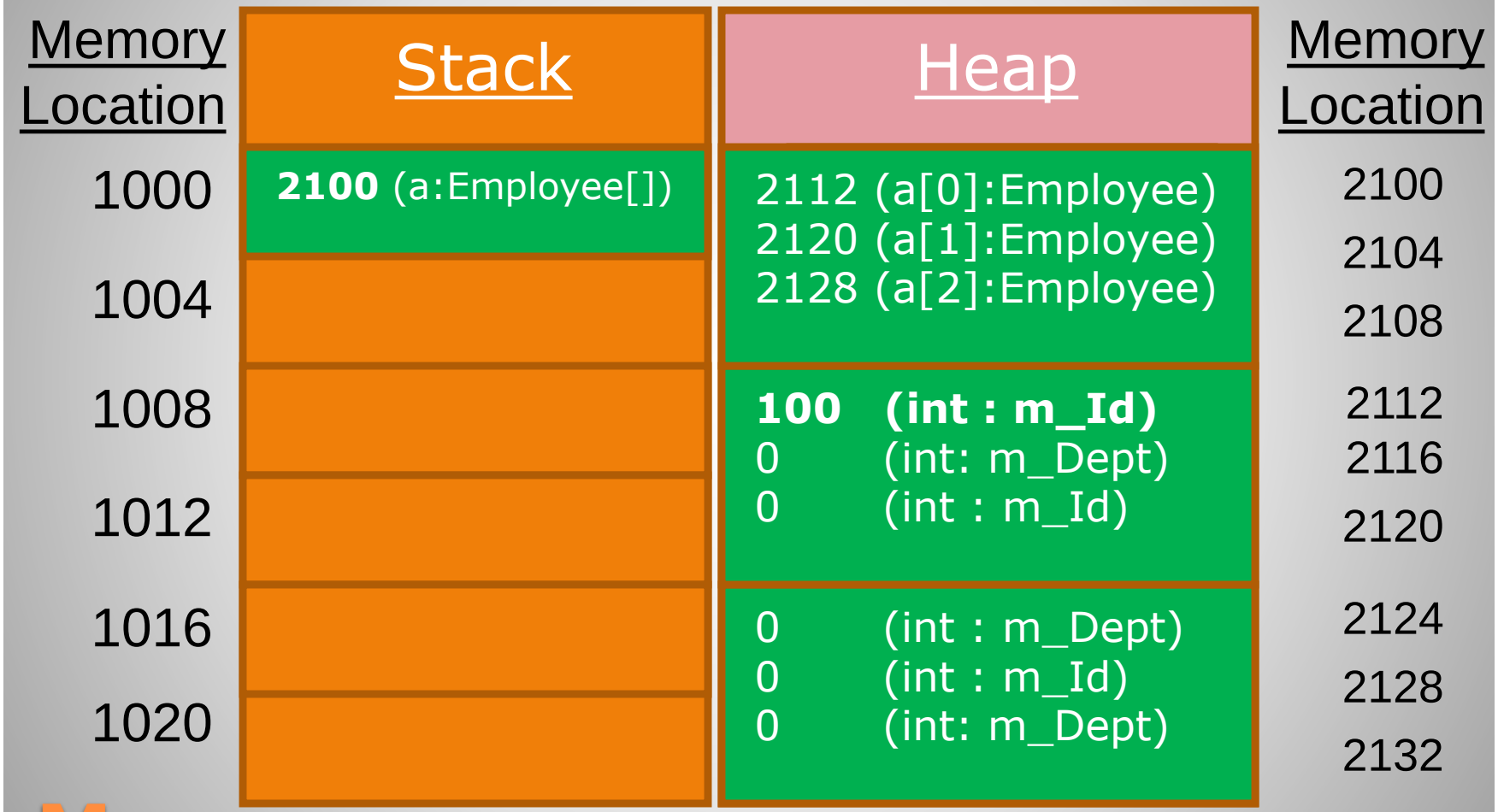
Call new for array

Call new for each element of the array

What will happen when this runs?

Memory

a[0].SetId(100); // Fine new was called on a[0]



Memory

Arrays of Reference Types

- Setting up an array of a reference type:
  1. Call new for the array itself. For example:

```
Employee[] a;                // Declare the array variable
a = new Employee[3];         // Call new to allocate the array
```

  2. Call new for EVERY element of the array. For example:

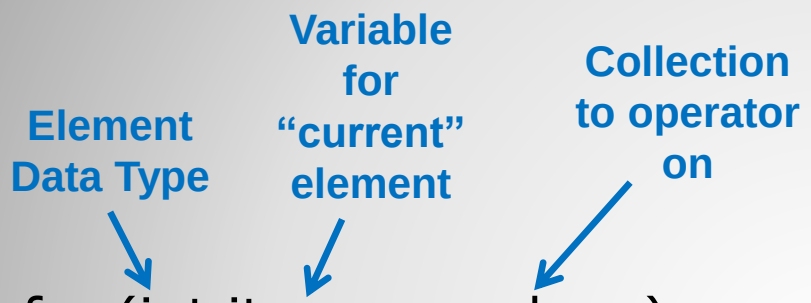
```
a[0] = new Employee(); // Allocate the first element of the array
a[1] = new Employee(); // Allocate the second element of the array
a[2] = new Employee(); // Allocate the third element of the array
```

for-each

- The for statement also has another form designed for iteration through Collections and arrays.
- This form is sometimes referred to as the ***enhanced for*** statement, and can be used to make your loops more compact and easy to read.
- More commonly referred to as a ***for-each***.
- Here is an example…
- Taken from:

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html

```
// Collection of data
int[] numbers = {1,2,3,4,5,6,7,8,9,10};
```



Every time through the loop the item variable will be filled with data from the next element in the collection

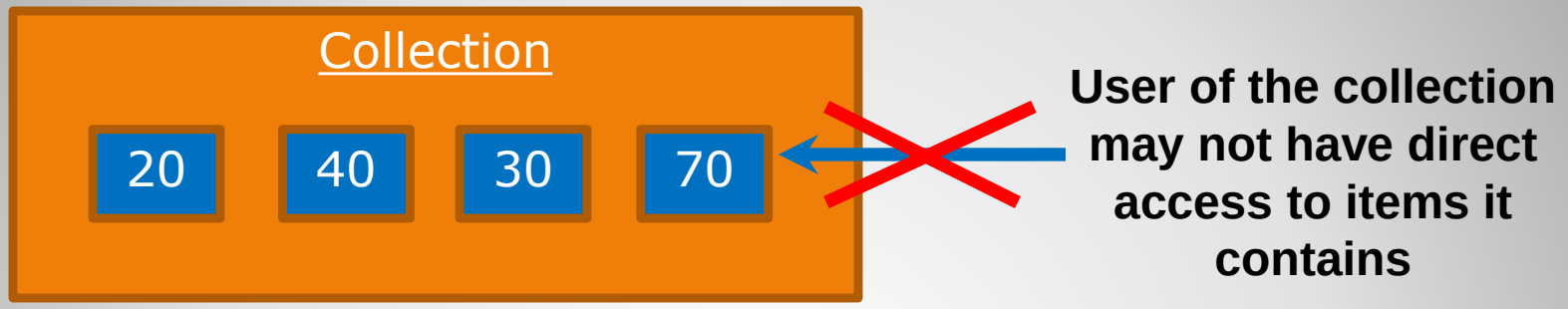
```
for (int item : numbers)
{
        System.out.println("Count is: " + item);
}
```

- Taken from:

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html

for-each

- Here is a collection with data (could be an array):

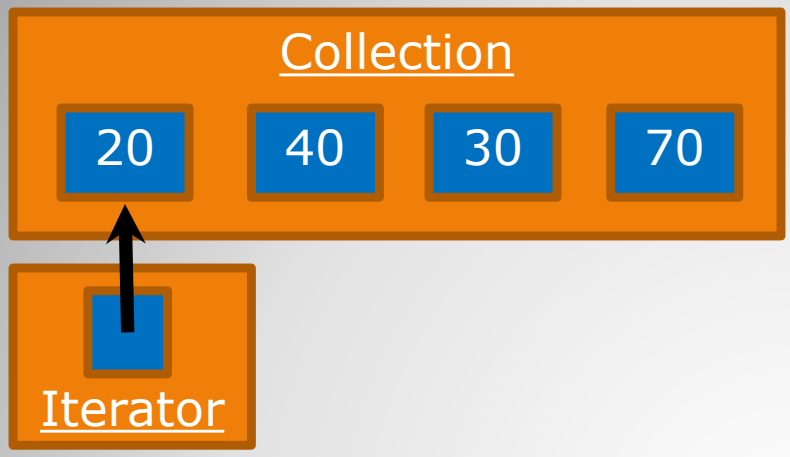


- Users of the collection may or may not have direct access to the items of the collection.
- There needs to be a way to "visit" each item of the collection while not having direct access to it.
- That is what an iterator is for.

Iterators

Iterators

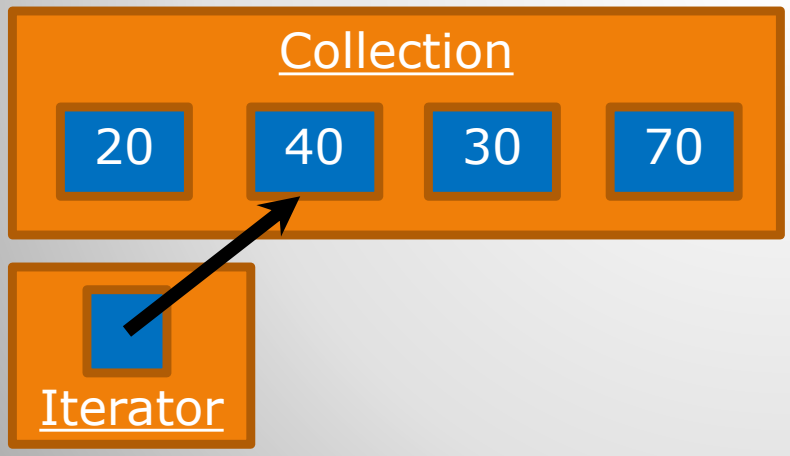
- Iterators are helper classes that have access to the items of the collection.
- An iterator points at one item of the class.
- In general, you can do the following with an iterator:
  - Get the data at that item.
  - Go to the next item in the collection.
  - Remove the item from that collection.
- For example…



This iterator points at the first item of the collection.

You can get the data (20) at that item if you want but not the other items.

If we told the iterator to go to the next item then it would look like the following….



Iterator now points at the second item.

You can get the data in the second item (40) but not the other items.

Iterators

- We will discuss creating and using iterators in chapter 10.

Iterators

class - Arrays

- Arrays (java.util.Arrays) – Predefined class in the Java API.
- Contains ***static*** methods that implement common array manipulations on normal Java arrays.
- For example:

```
double[] da = { 8.4, 9.3, 0.2, 7.9, 3.4 }
Arrays.sort( da ); // Pass array into sort method, sorts array

// Prints array in sorted order:  0.2   3.4   7.9   8.4   9.3
for (double value : da) {
        System.out.printf( "%.1f  ", value);
}
```

- Other Arrays static methods: binarySearch, equals, fill, arraycopy

class - ArrayList

- Predefined data structure in Java API.
- Collection that stores its objects just like a normal array.
- Put values in and get values out of the collection using an index.
- An ArrayList can **resize** itself to accommodate more elements.
- **Only stores nonprimitive types (cannot be used to store int, double, etc…).**

class - ArrayList

- ArrayList is a "generic" type (similar to templates in C++)
- Can dynamically change its size to accommodate more elements (a normal Java array cannot resize). For example:

```
ArrayList<String> al;
al = new ArrayList<String>();
```

Must indicate the data type of elements that will be stored in each instance

```
al.add( "red");          // Adds “red” to end of ArrayList
al.add( 0, "yellow");    // Adds “yellow at index 0
String s = al.get(0);    // Gets element at index 0
al.clear();              // Removes all elements
```

- Textbook describes other methods such as: contains, indexOf, remove, size etc…

Review

- Is an array a primitive or reference type?
- What are the valid indexes for a 100 element array?
- Is an array that is declared in a method stored on the stack or the heap or both?
- How many calls to new would be necessary to create an array of 5 reference type elements?

- End of Slides.

End of Slides