

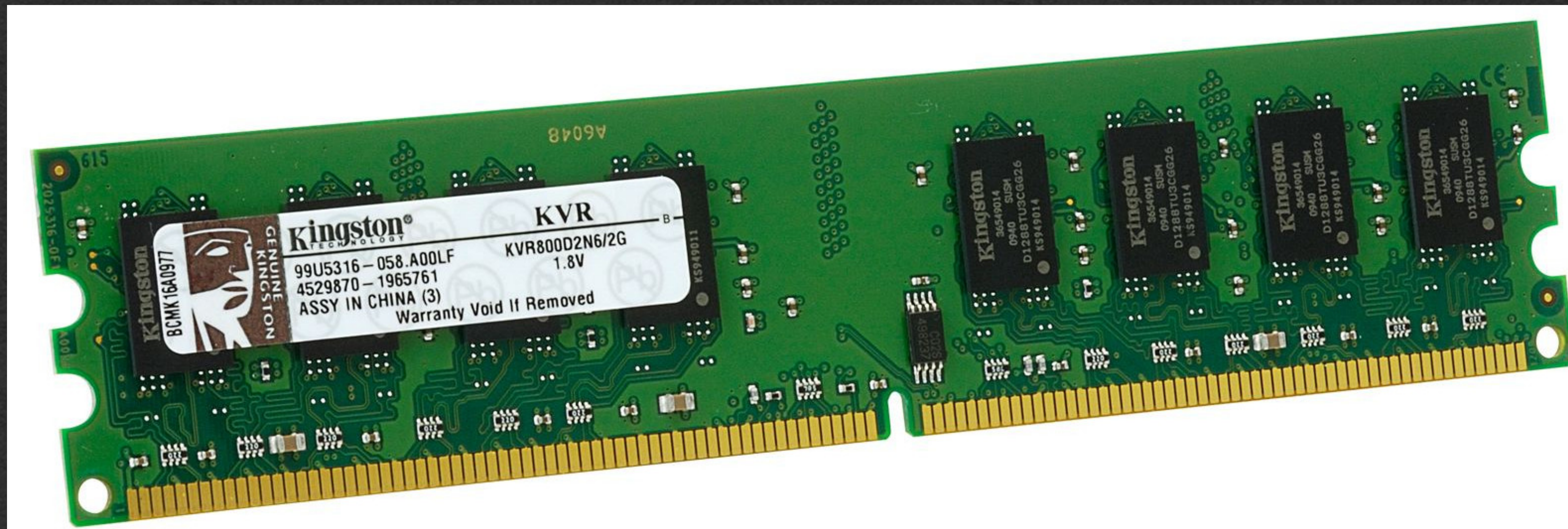
Java

ArrayList, HashMap

Memory

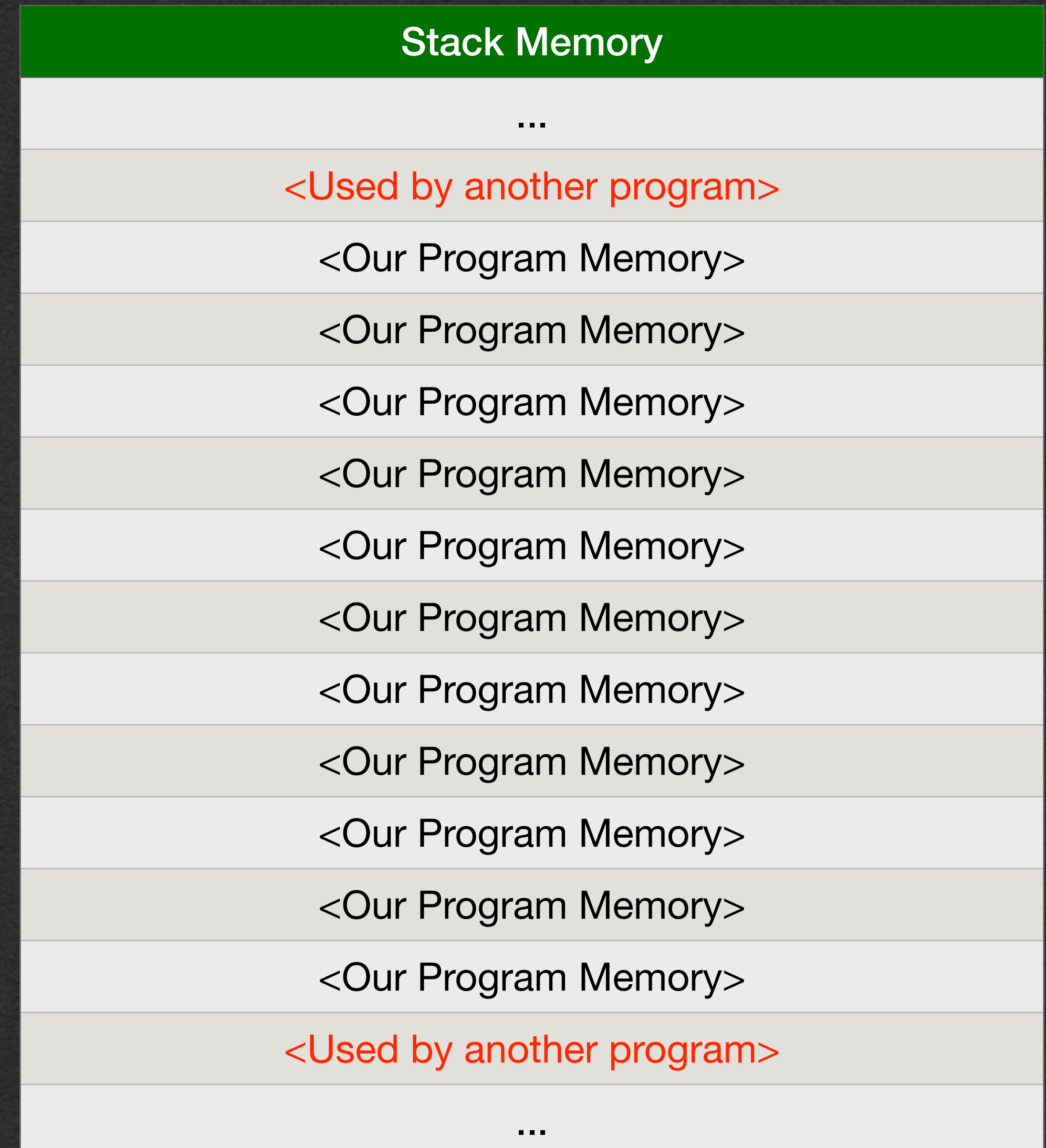
Let's Talk About Memory

- Random Access Memory (RAM)
- Access any value by index
- Effectively, a giant array
- All values in your program are stored here



Let's Talk About Memory

- Operating System (OS) controls memory
- On program start, OS allocates a section of memory for our program
- Gives access to a range of memory addresses/indices



Stack Memory

- Fixed section of memory used to store variables and stack frames
- One continuous section of RAM
- LIFO - Last In First Out
 - New values are added to the end of the stack
 - Only frames at the end of the stack can be removed

Heap Memory

- ArrayLists and HashMaps will be stored in heap memory
- Heap memory is dynamic
 - We can "ask" the OS/JVM for more heap space as needed
- Heap memory can be anywhere in RAM
 - Location is not important
 - Location can change
- Use **references** to find data
 - **Variables only store references to values in the heap**

ArrayList

Java - ArrayList

```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```

- Similar to:
 - List in Python
 - Array in JavaScript
- Sequential data structure
 - Order matters
- Values indexed starting at 0

Java - ArrayList

```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```

- ArrayList is built-in with Java
- However, it is not automatically available
- Unlike String, int, double, etc.
- Must import ArrayList
- The ArrayList class is in the java.util package
- Importing makes the class available in your code

Java - ArrayList

```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```

- Use the "new" keyword to create a new ArrayList
- Must have <> which is a type parameter list
- Can also have <Integer> in this example
- Must have () which is an empty argument list
- This calls the classes *constructor* method and returns an *object*
- Much more detail to come in week 4

Java - ArrayList

```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```

- An ArrayList variable should have a type parameter in <>
- This ArrayList has a type parameter of Integer
- We say this is an "ArrayList of Integers"
- This ArrayList can **only ever** store Integers

Java - ArrayList

```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```

- The type parameter has to be a class
- Class types start with capital letters
- You cannot create an ArrayList of ints

Java - ArrayList

```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```

- `int` \approx `Integer`
- `double` \approx `Double`
- `boolean` \approx `Boolean`
- Use the class equivalents for our primitive (starts with lowercase letter) types
- [In most cases] Java will automatically convert between the two
- Conversion is called auto-boxing
- We'll always use the primitive types unless we must use the class equivalent

Java - ArrayList

```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```

- Call the add method to insert a value at the end of the ArrayList
- Call get with an index to retrieve that value at that index
- **Cannot** use [index] to access a value in an ArrayList

Memory Diagram


```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }
    ➡ public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```

Stack		Heap
Name	Value	
		<u>in/out</u>

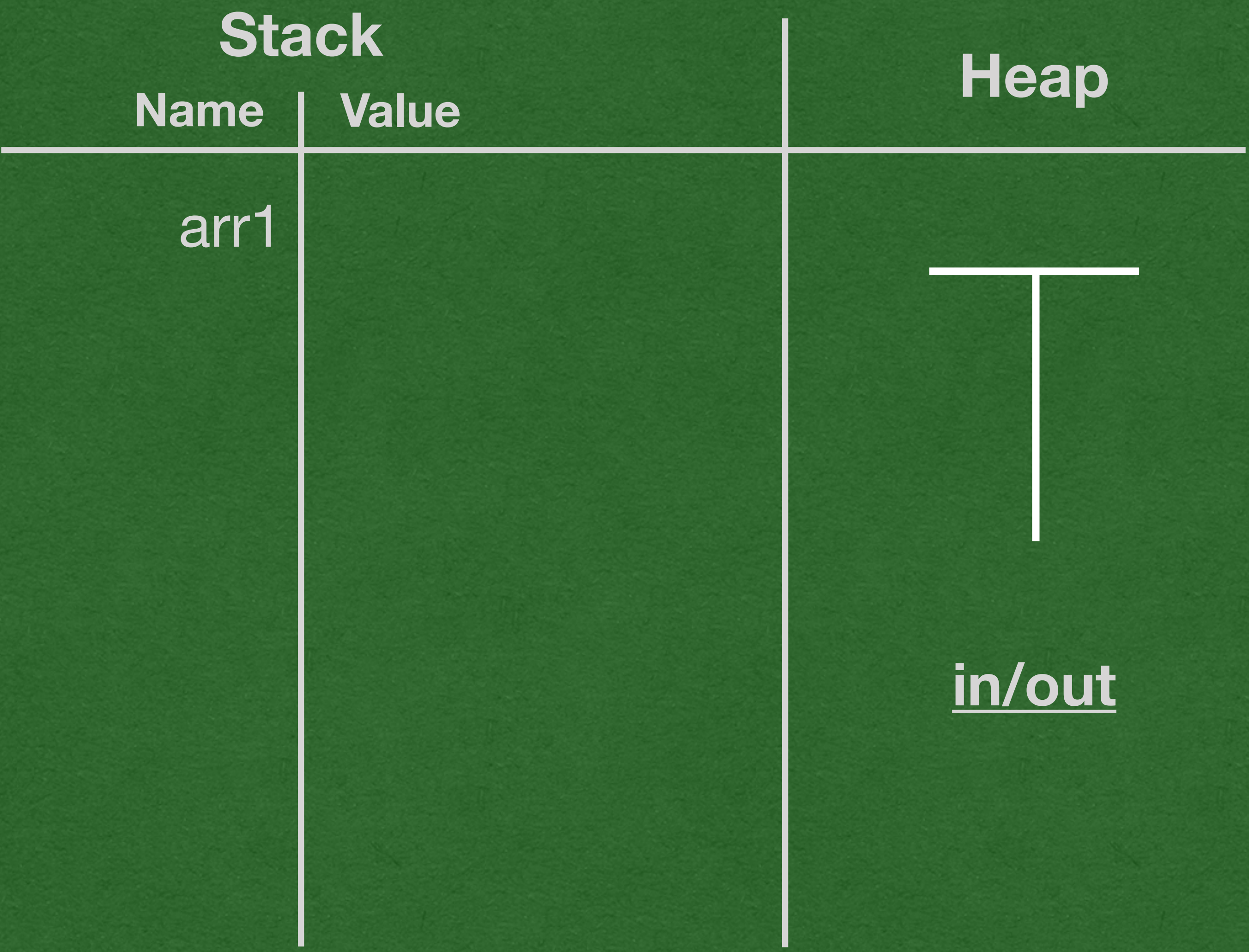
- It all starts the same
- It will quickly become very different


```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ➡ ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```



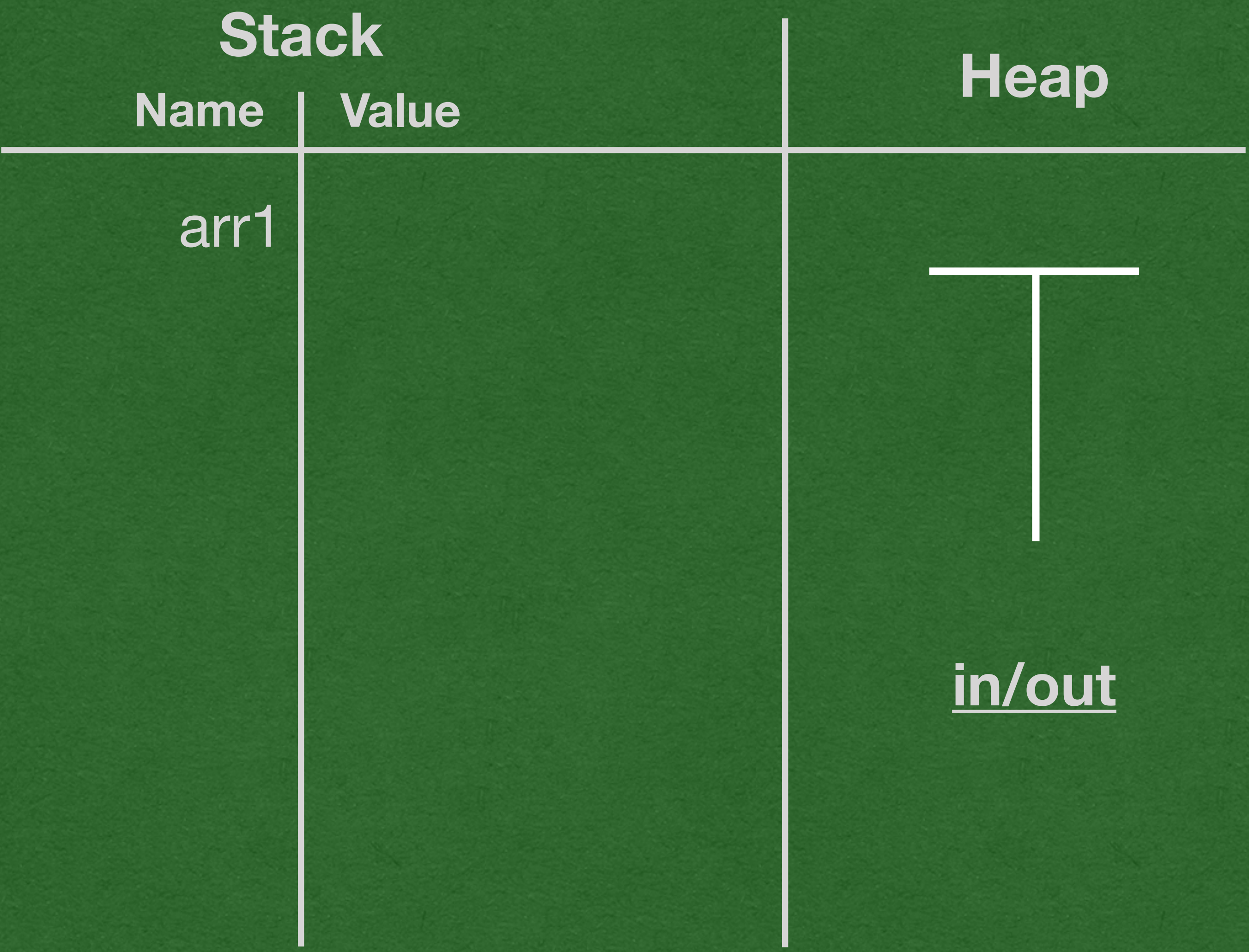
- We create an ArrayList
- ArrayLists go in the heap!


```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ➡ ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```



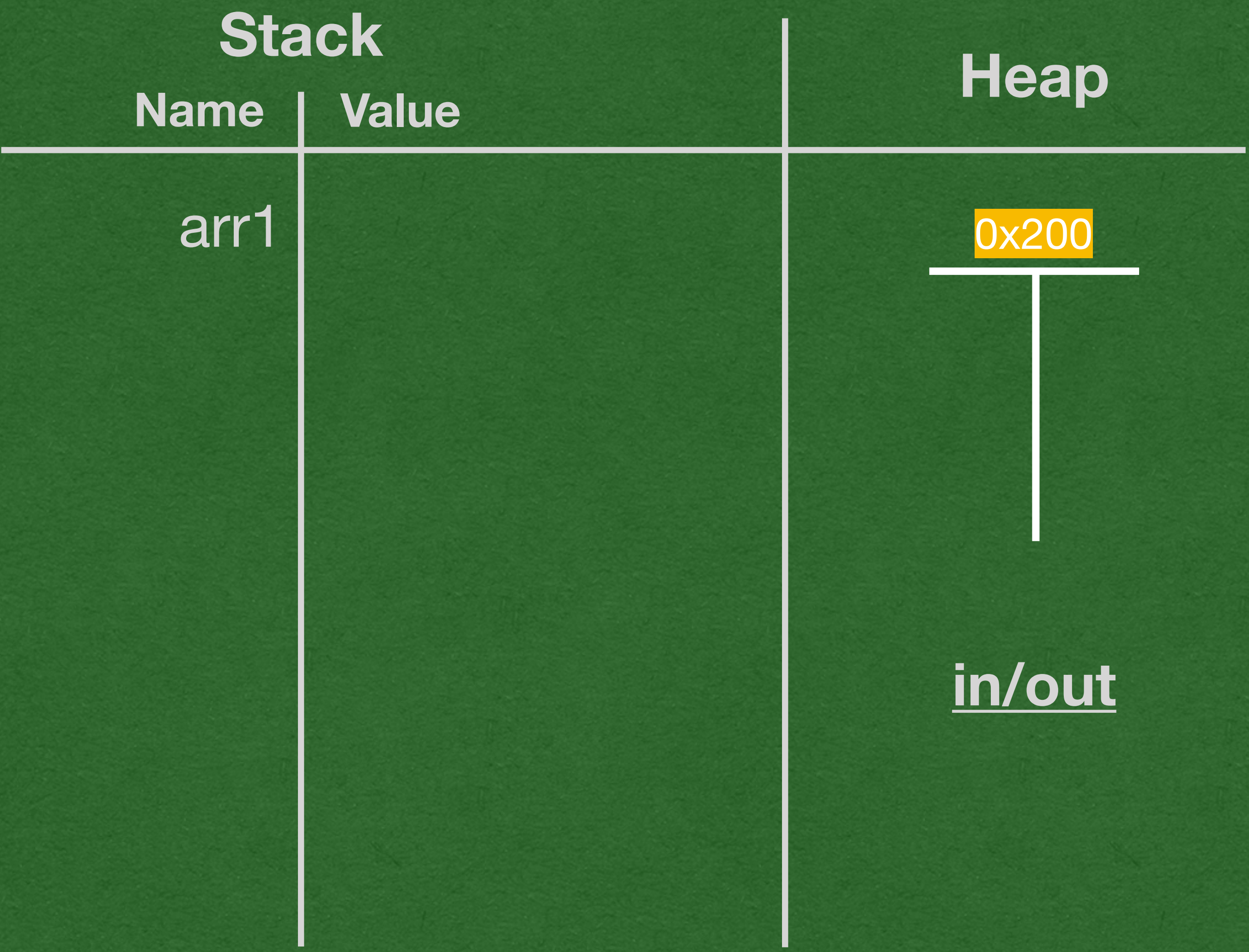
- When an ArrayList is created on the heap:
- Create 2 columns: One for indices, one for values


```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ➡ ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```



- Value on the heap always get a memory address
- "0x" followed by a number (You can choose any numbers for your diagrams)
- This tells java where in memory it can find the value


```

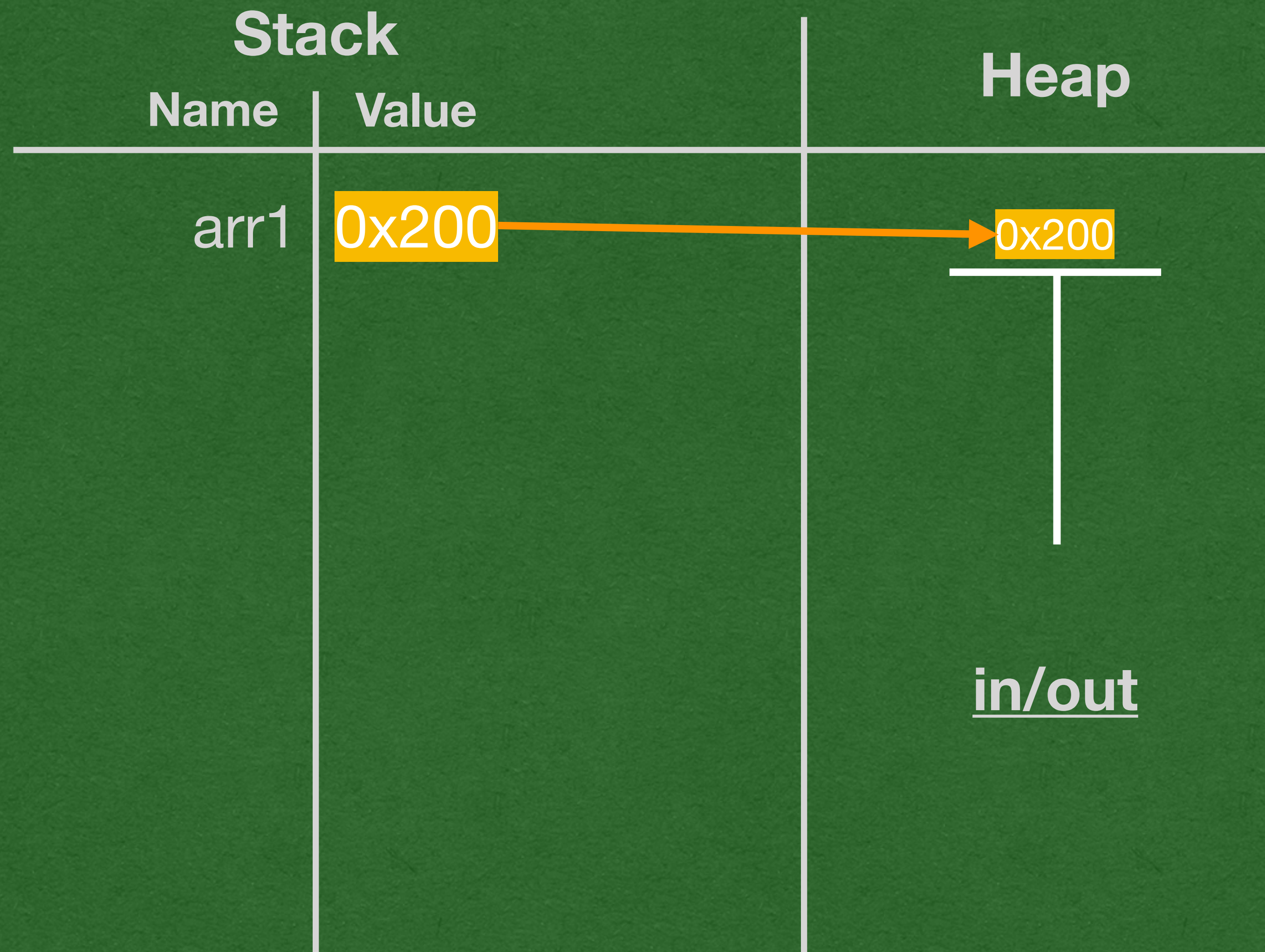
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ➡ ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}

```



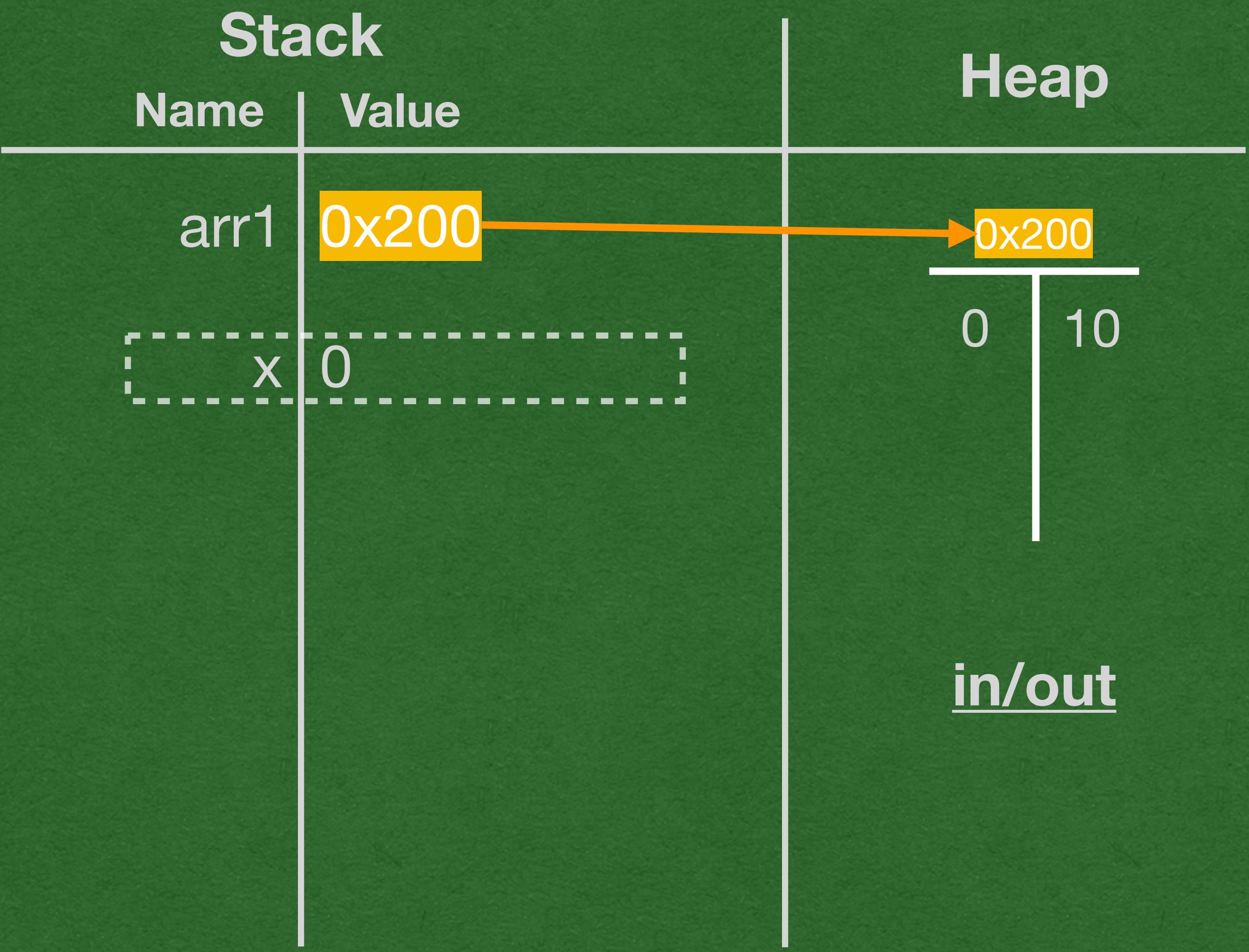
- When a variable "stores" a value that's on the heap, it only store a **reference** to that value
- arr1 only stores instructions of how to find the ArrayList in the heap


```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        ➡ for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```



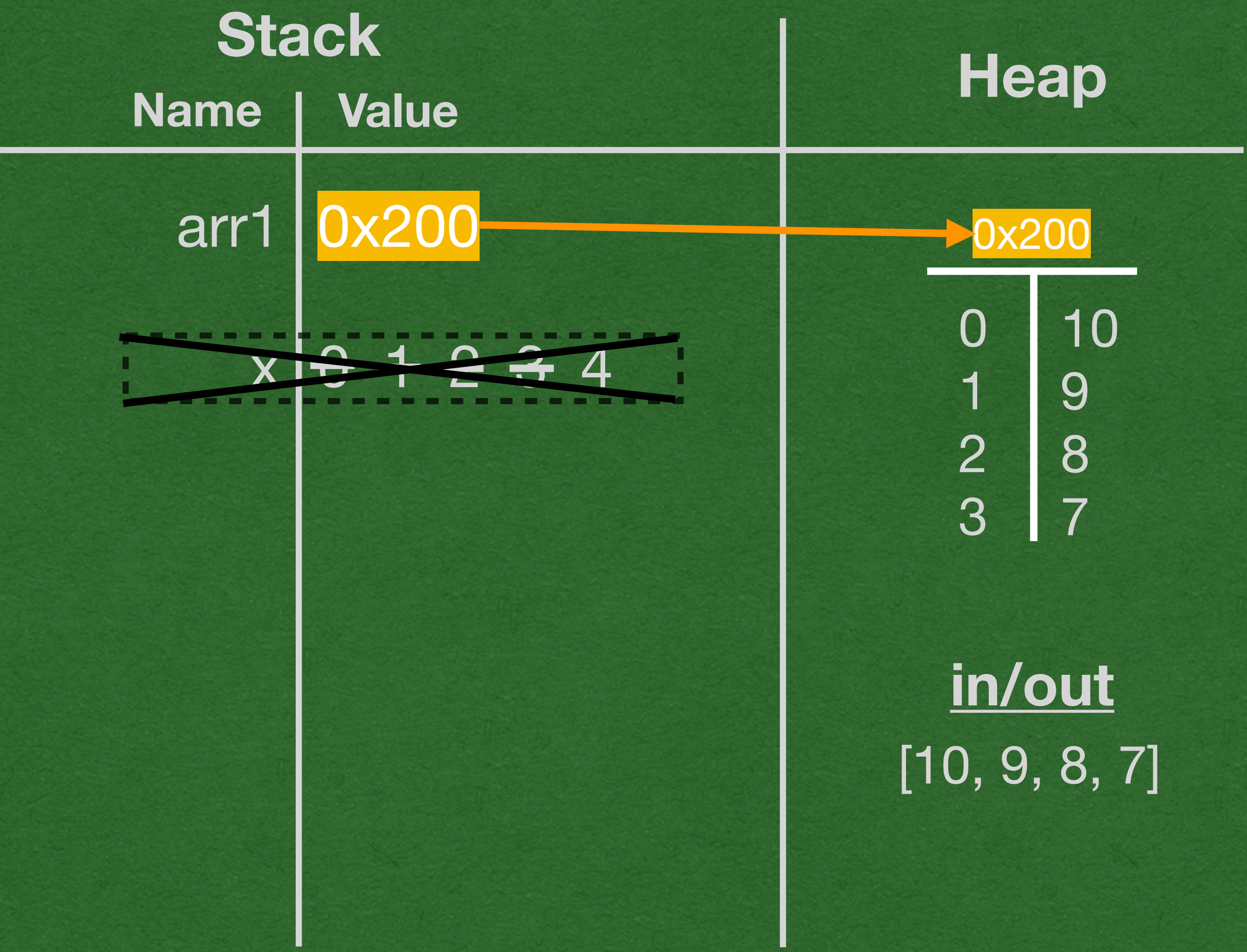
- Each time we add a value to an ArrayList, it is added to the next index


```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        ➡ System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```



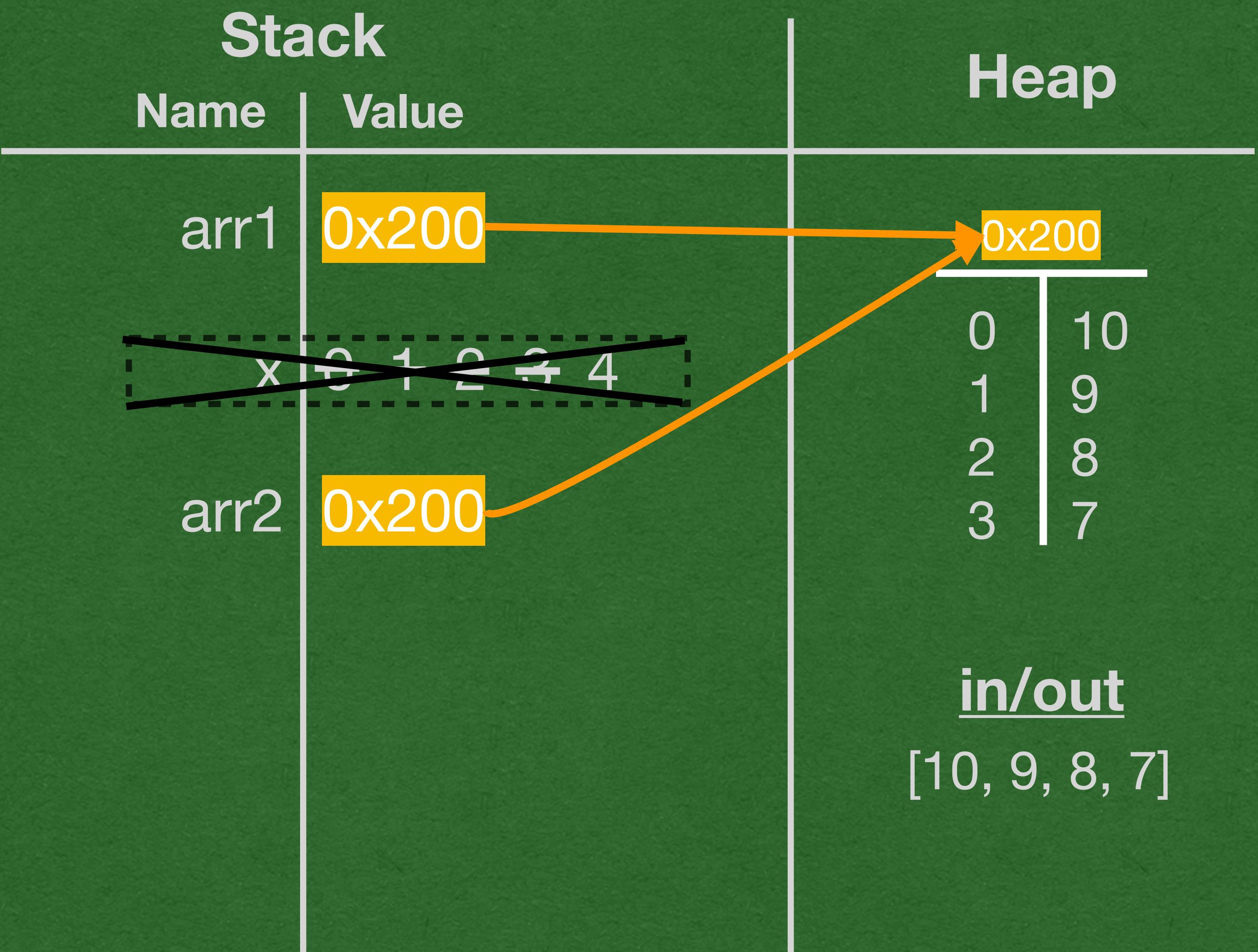
- Printing an ArrayList will print all it's values in [] separated by commas


```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ➡ ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```



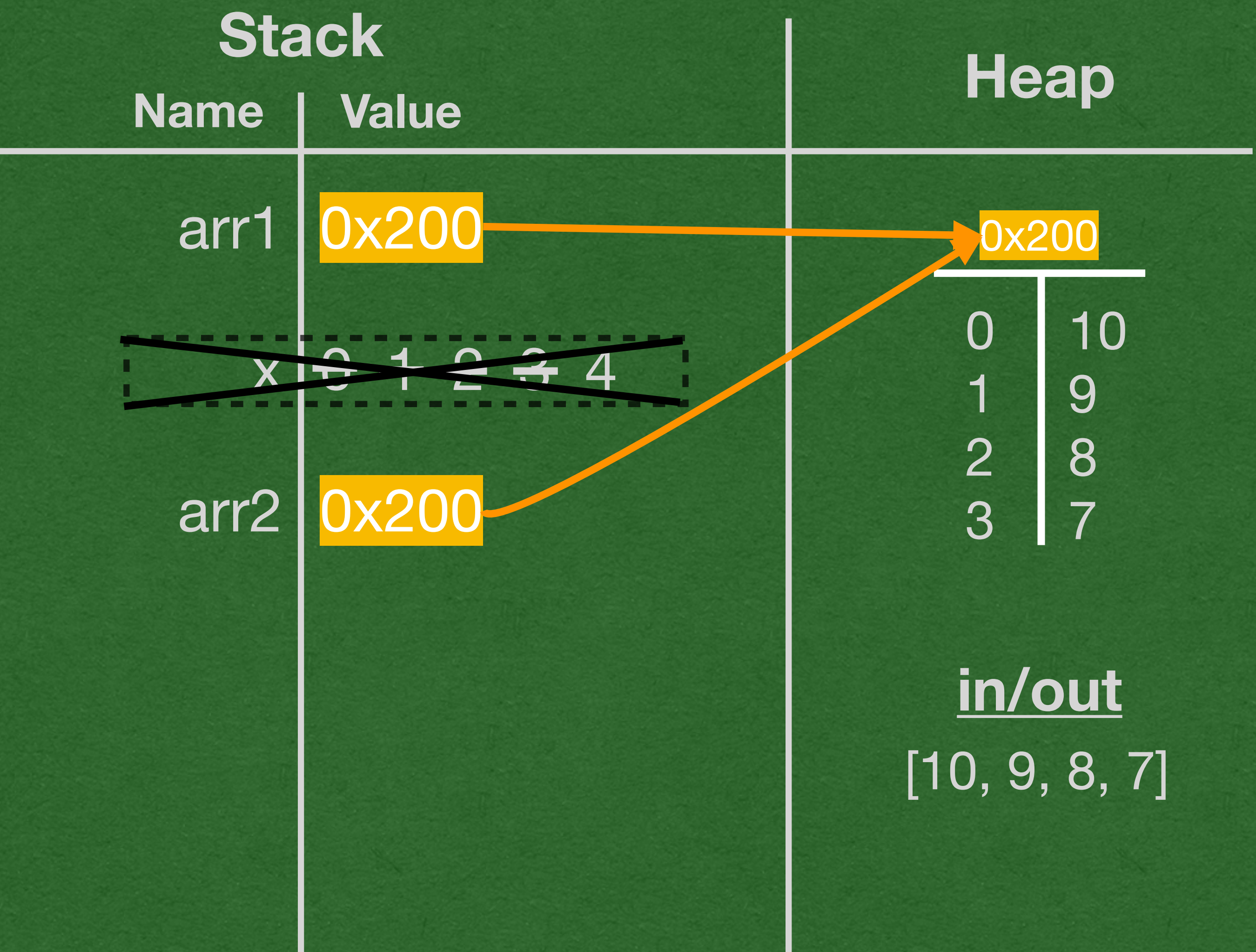
- When a variable is assigned a value that is a reference, **only** the **reference** is assigned!
- There is no copy of the ArrayList created. Only 1 ArrayList exists in memory
- That ArrayList is *referred to* by the 2 variables that store its reference


```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ➡ ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```



- This is ****assign-by-reference****
- Only the reference is assigned

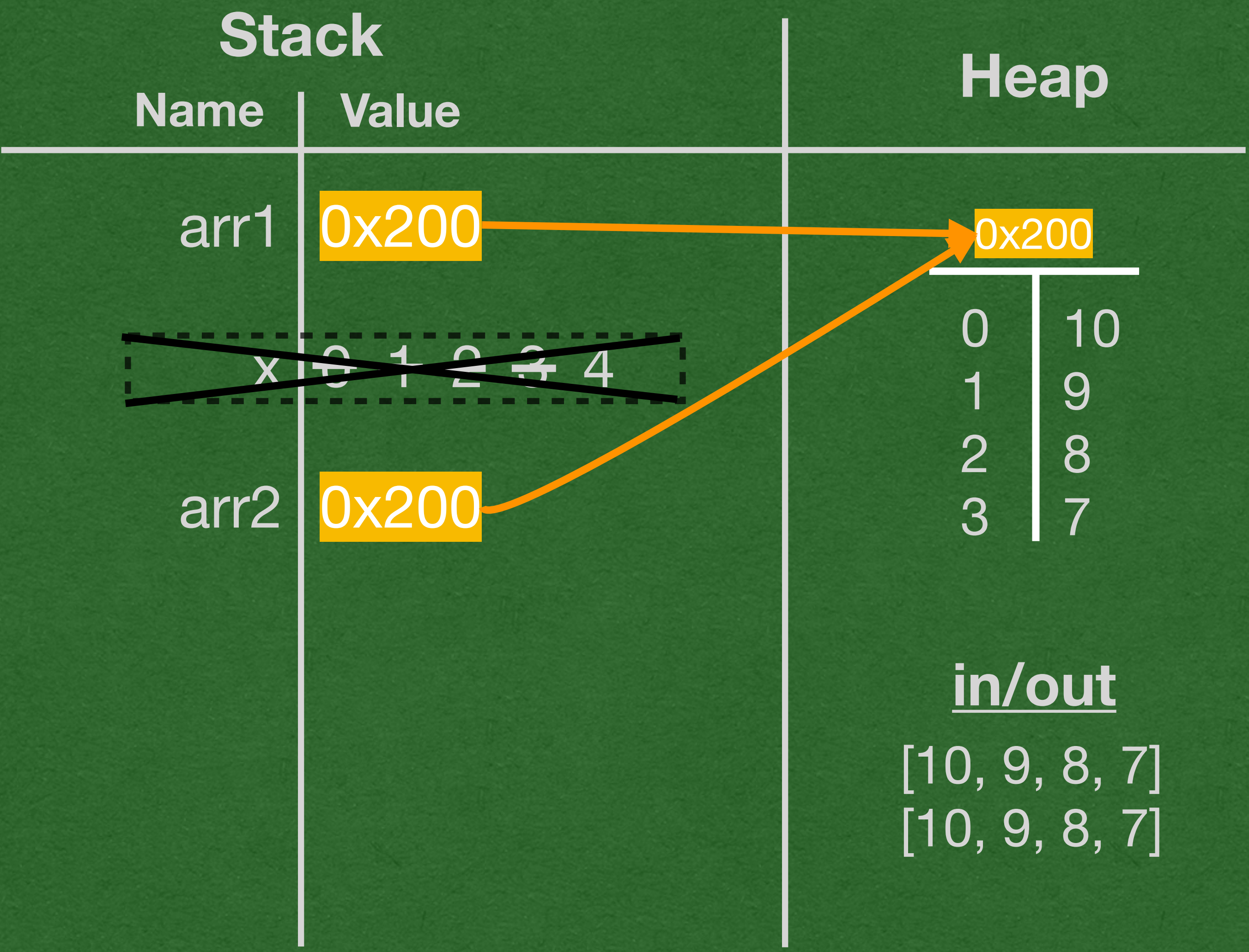
Technically it's assign-by-value, but the value is a reference


```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        ➡ System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```



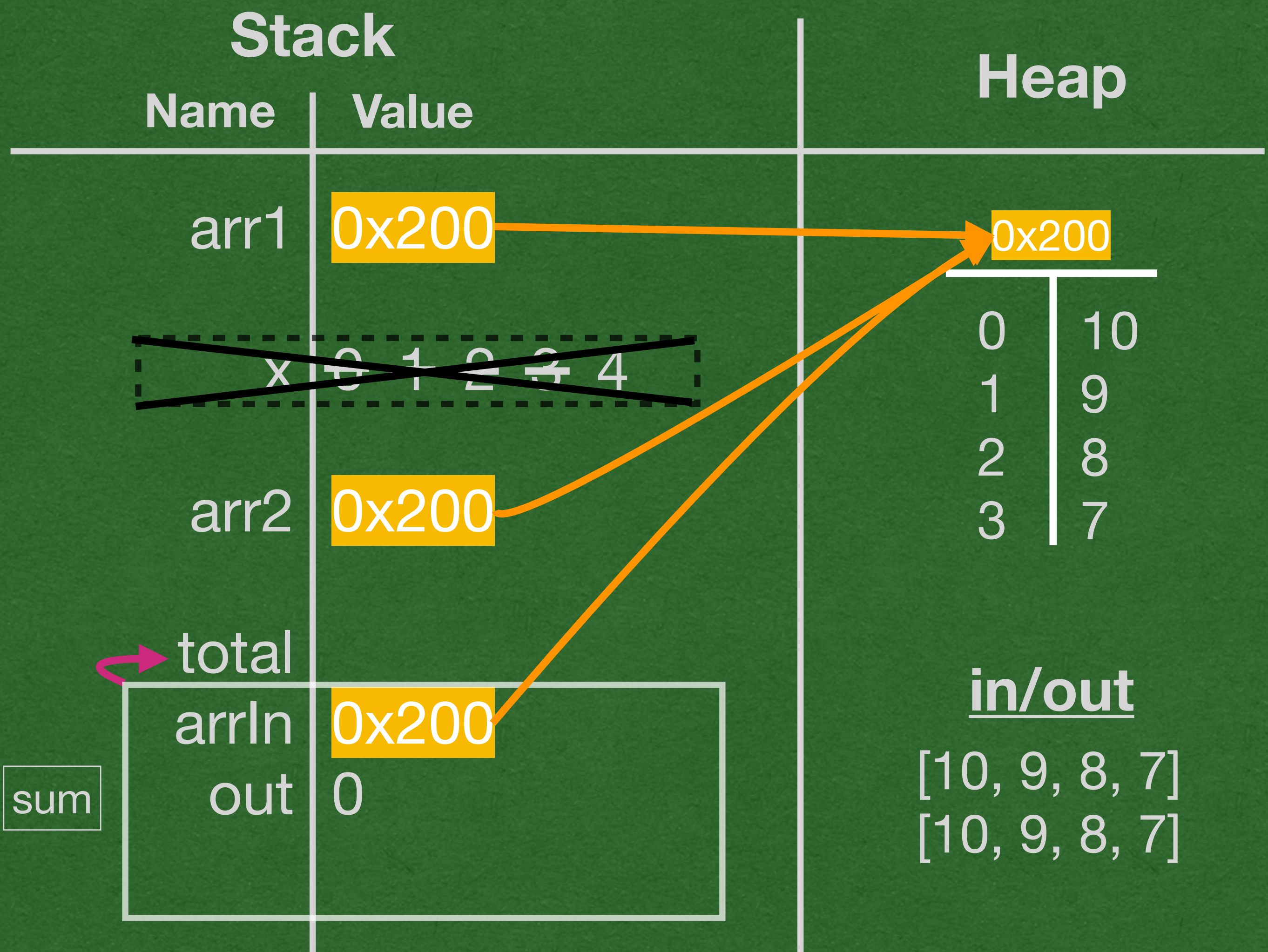
- arr2 refers to the same ArrayList as arr1 -- the only ArrayList in this example
- Printing arr2 is the same as printing arr1


```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        ➡ int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        ➡ int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```



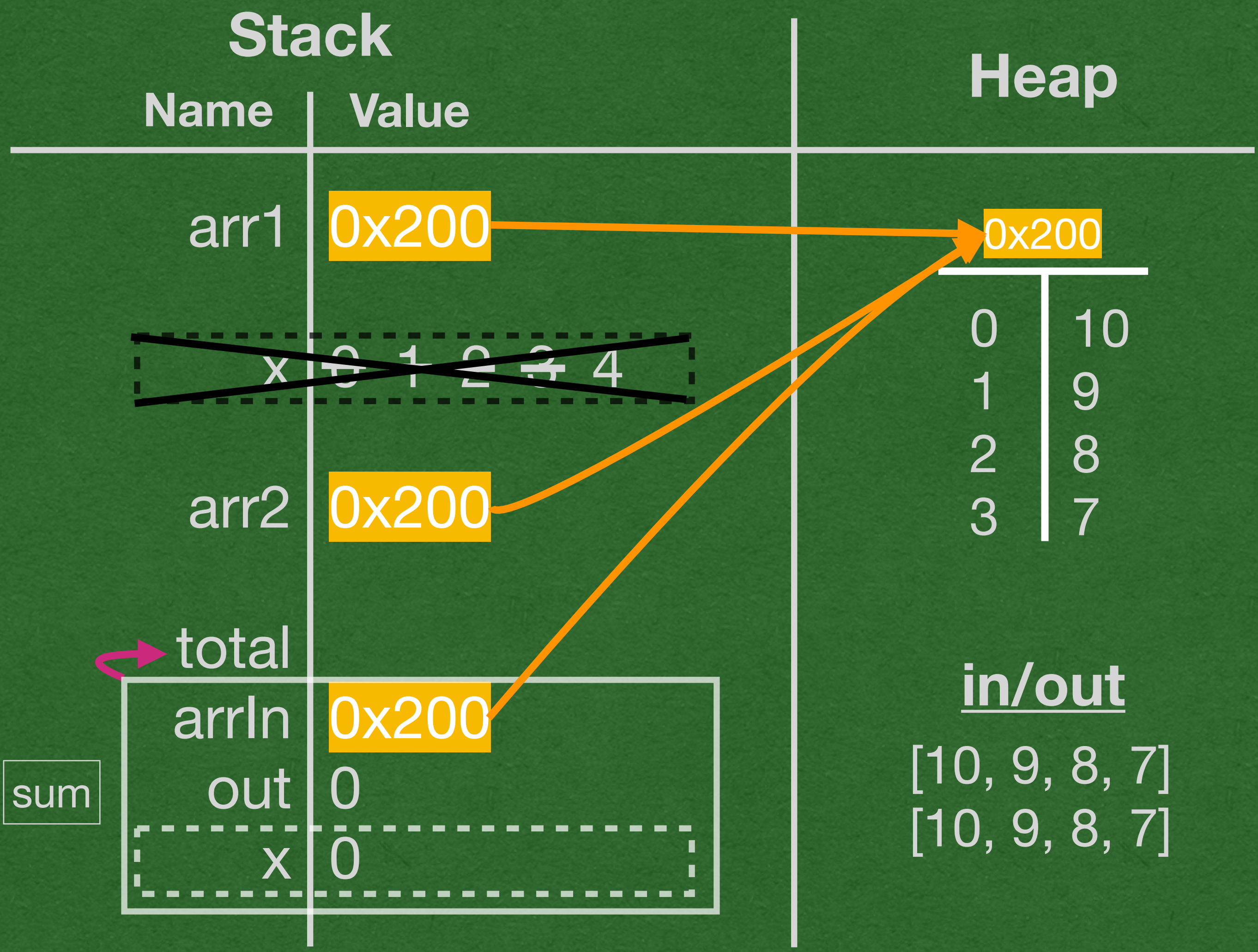
- When a method is called that take an object on the heap as a parameter, only the reference is passed into the stack frame
 - This is ****pass-by-reference****
- Technically it's pass-by-value, but the value passed is a reference


```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        ➡ for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        ➡ int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```



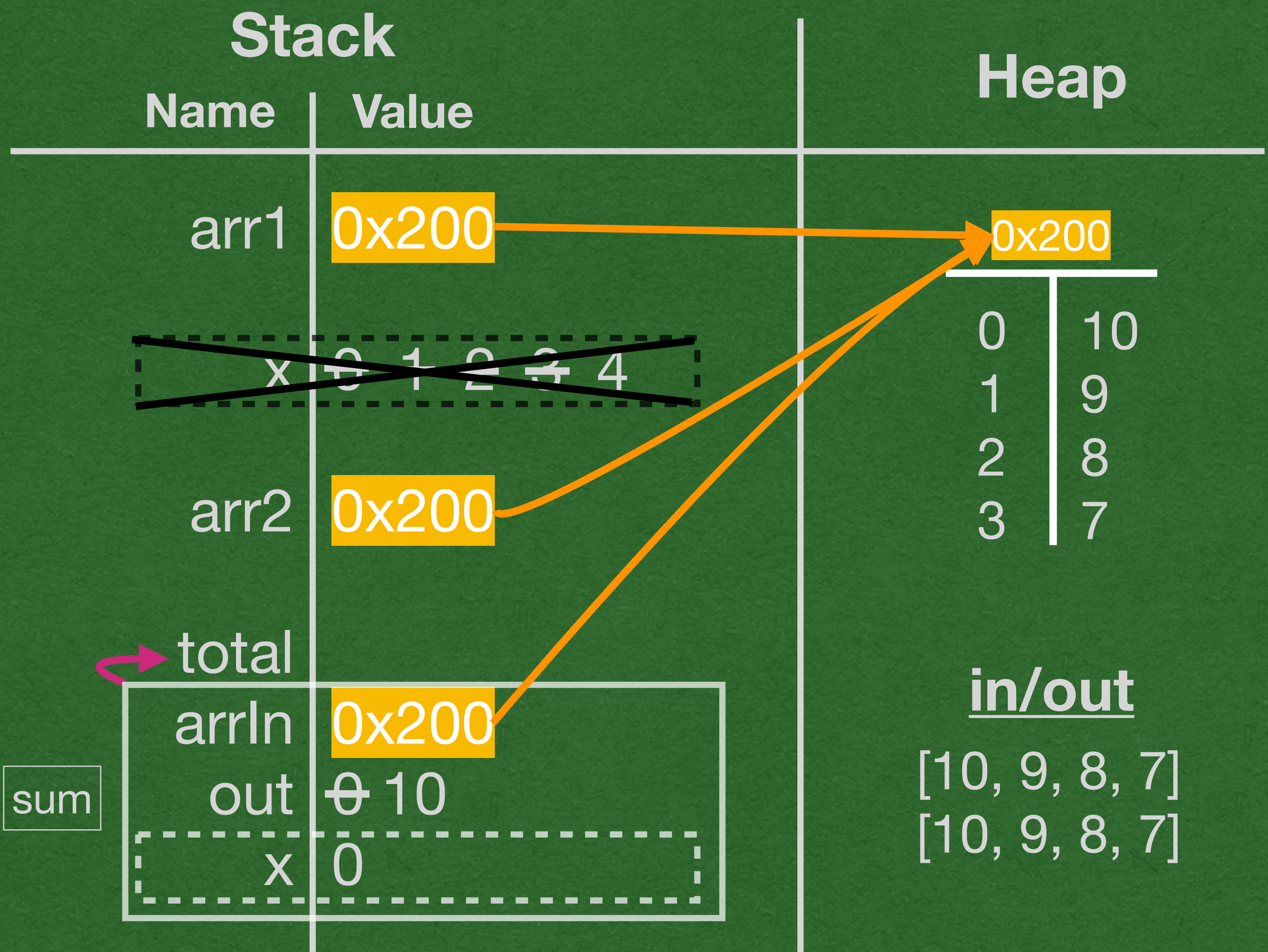
- When using the reference, the dot operator . means we follow the reference to the object to which it refers
- `arrIn.size()` means - go to the ArrayList referred to by this reference and call it's size method


```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```



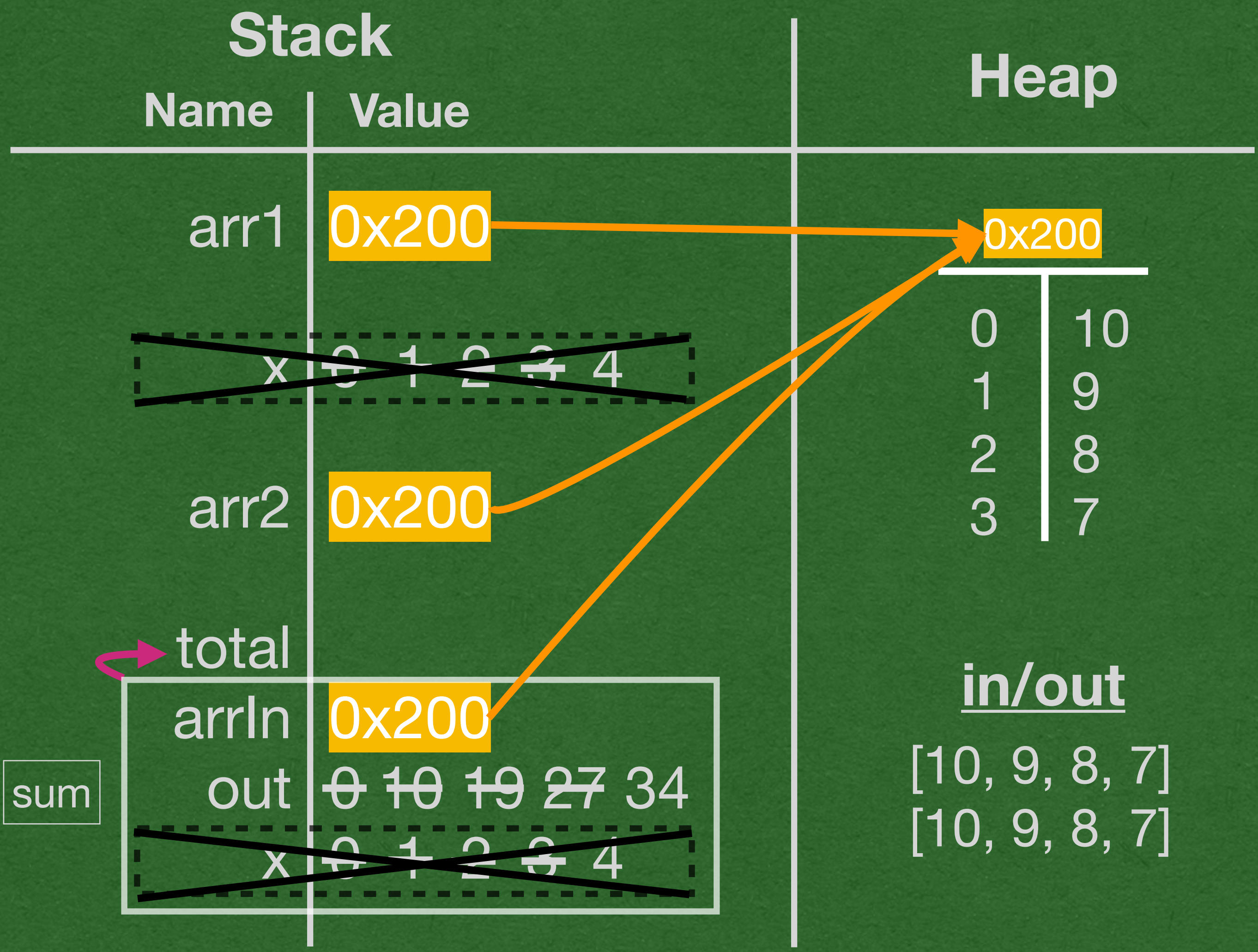
- `arrIn.get(x)`
- Follow the reference
- Return the value stored at index `x` and add 10 to the `out` variable


```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```



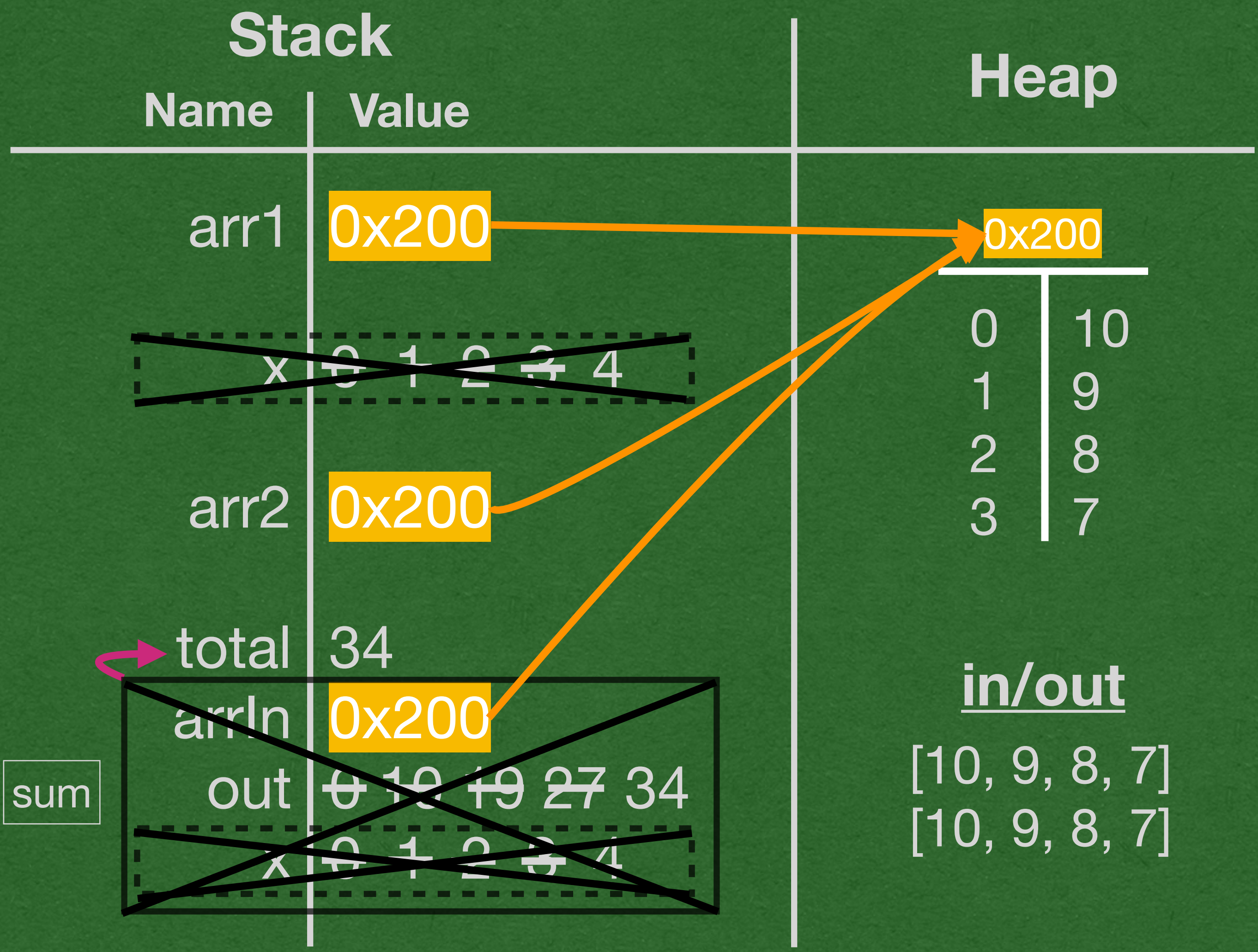
- When x is 4, `x<arrIn.size()` is false
- The loop ends and x is removed from memory


```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```



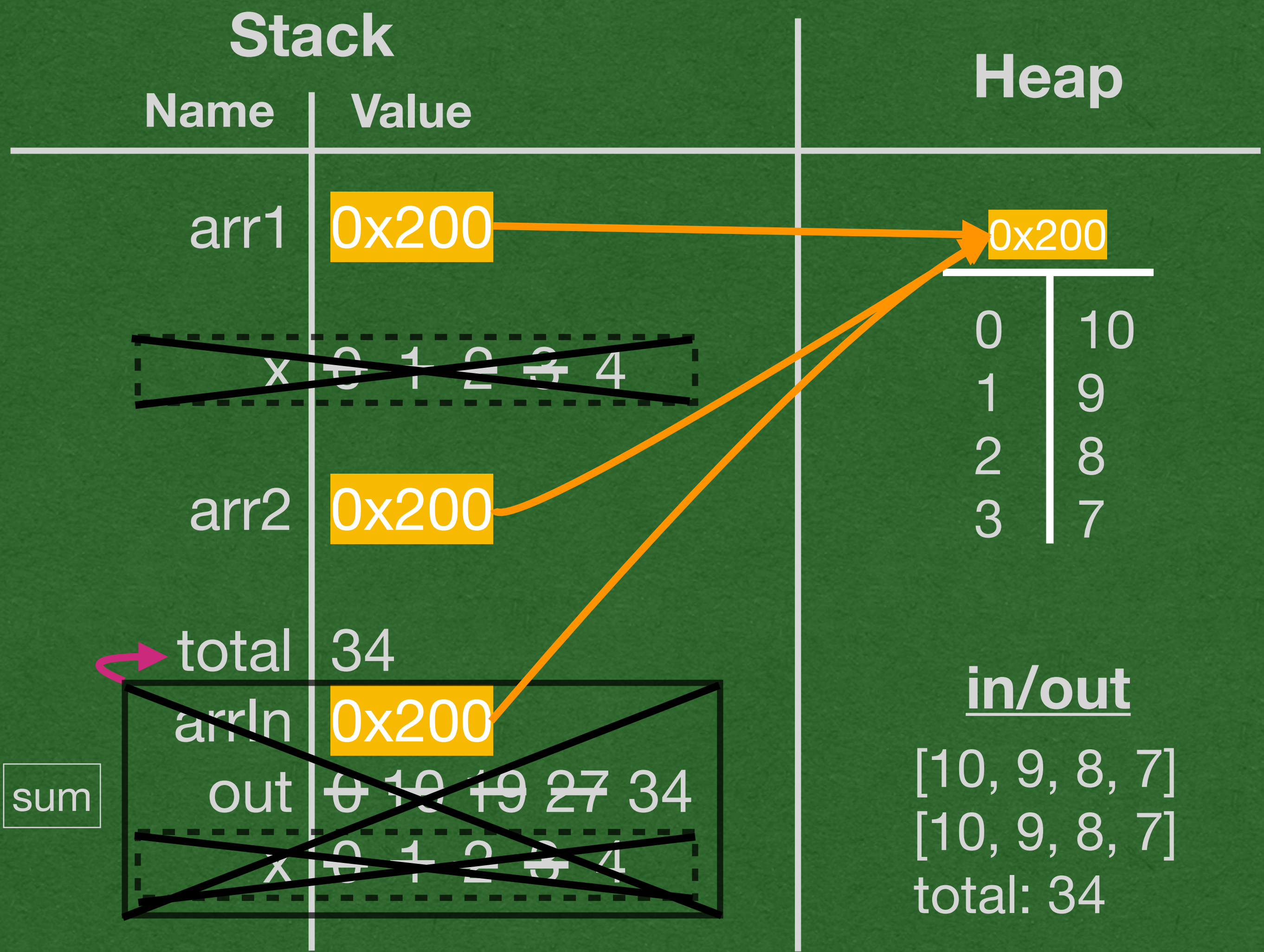
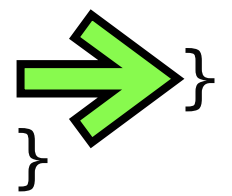
- Return the value of the out variable to the total variable
- The entire stack frame is removed from memory


```
package week2;

import java.util.ArrayList;

public class ArrayList1 {
    public static int sum(ArrayList<Integer> arrIn) {
        int out = 0;
        for (int x=0; x<arrIn.size(); x++) {
            out += arrIn.get(x);
        }
        return out;
    }

    public static void main(String[] args) {
        ArrayList<Integer> arr1 = new ArrayList<>();
        for (int x=0; x<4; x++) {
            arr1.add(10-x);
        }
        System.out.println(arr1);
        ArrayList<Integer> arr2 = arr1;
        System.out.println(arr2);
        int total = sum(arr1);
        System.out.println("total: " + total);
    }
}
```



- Print total
- End of program

HashMap

Java - HashMap

```
package week2;

import java.util.HashMap;

public class HashMap1 {
    public static void main(String[] args) {
        HashMap<String, Integer> bills = new HashMap<>();

        bills.put("Allen", 17);
        bills.put("Diggs", 14);
        System.out.print("What is Allen's number? ");
        System.out.println(bills.get("Allen"));
        System.out.println(bills);
        for (String key : bills.keySet()) {
            System.out.println(key);
        }
        for (Integer value : bills.values()) {
            System.out.println(value);
        }
        for (String key : bills.keySet()) {
            int value = bills.get(key);
            System.out.print(key + "'s number is: ");
            System.out.println(value);
        }
    }
}
```

- Similar to:
 - Dictionary in Python
 - Object in JavaScript
- Key-Value Store
 - Order does not matter
 - Cannot have duplicate keys
- Used to associate keys with values

Java - HashMap

```
package week2;

import java.util.HashMap;

public class HashMap1 {
    public static void main(String[] args) {
        HashMap<String, Integer> bills = new HashMap<>();

        bills.put("Allen", 17);
        bills.put("Diggs", 14);
        System.out.print("What is Allen's number? ");
        System.out.println(bills.get("Allen"));
        System.out.println(bills);
        for (String key : bills.keySet()) {
            System.out.println(key);
        }
        for (Integer value : bills.values()) {
            System.out.println(value);
        }
        for (String key : bills.keySet()) {
            int value = bills.get(key);
            System.out.print(key + "'s number is: ");
            System.out.println(value);
        }
    }
}
```

- Must import before use
- Most types we use from here onward need to be imported
- Only primitives and classes in the java.lang package do not need to be imported

Java - HashMap

```
package week2;

import java.util.HashMap;

public class HashMap1 {
    public static void main(String[] args) {
        HashMap<String, Integer> bills = new HashMap<>();

        bills.put("Allen", 17);
        bills.put("Diggs", 14);
        System.out.print("What is Allen's number? ");
        System.out.println(bills.get("Allen"));
        System.out.println(bills);
        for (String key : bills.keySet()) {
            System.out.println(key);
        }
        for (Integer value : bills.values()) {
            System.out.println(value);
        }
        for (String key : bills.keySet()) {
            int value = bills.get(key);
            System.out.print(key + "'s number is: ");
            System.out.println(value);
        }
    }
}
```

- HashMaps have 2 type parameters
- First is the type of the keys
- Second is the type of the values
- We say this is a:
 - HashMap from String to Integer
 - Maps Strings to Integers

Java - HashMap

```
package week2;

import java.util.HashMap;

public class HashMap1 {
    public static void main(String[] args) {
        HashMap<String, Integer> bills = new HashMap<>();

        bills.put("Allen", 17);
        bills.put("Diggs", 14);
        System.out.print("What is Allen's number? ");
        System.out.println(bills.get("Allen"));
        System.out.println(bills);
        for (String key : bills.keySet()) {
            System.out.println(key);
        }
        for (Integer value : bills.values()) {
            System.out.println(value);
        }
        for (String key : bills.keySet()) {
            int value = bills.get(key);
            System.out.print(key + "'s number is: ");
            System.out.println(value);
        }
    }
}
```

- Add key-value pairs using "put"
- Retrieve a value at a particular key using "get"

Java - HashMap

```
package week2;

import java.util.HashMap;

public class HashMap1 {
    public static void main(String[] args) {
        HashMap<String, Integer> bills = new HashMap<>();

        bills.put("Allen", 17);
        bills.put("Diggs", 14);
        System.out.print("What is Allen's number? ");
        System.out.println(bills.get("Allen"));
        System.out.println(bills);
        for (String key : bills.keySet()) {
            System.out.println(key);
        }
        for (Integer value : bills.values()) {
            System.out.println(value);
        }
        for (String key : bills.keySet()) {
            int value = bills.get(key);
            System.out.print(key + "'s number is: ");
            System.out.println(value);
        }
    }
}
```

- for-each loop
- Or "enhanced" loop in the world of Java
- Very similar to Python loops
for (type variableName : dataStructure)
- Read: for variableName in dataStructure

Doesn't have to be a data structure.
Anything that can be iterated over will work

Java - HashMap

```
package week2;

import java.util.HashMap;

public class HashMap1 {
    public static void main(String[] args) {
        HashMap<String, Integer> bills = new HashMap<>();

        bills.put("Allen", 17);
        bills.put("Diggs", 14);
        System.out.print("What is Allen's number? ");
        System.out.println(bills.get("Allen"));
        System.out.println(bills);
        for (String key : bills.keySet()) {
            System.out.println(key);
        }
        for (Integer value : bills.values()) {
            System.out.println(value);
        }
        for (String key : bills.keySet()) {
            int value = bills.get(key);
            System.out.print(key + "'s number is: ");
            System.out.println(value);
        }
    }
}
```

- keySet
 - Allows us iterate (loop) over the keys
- values
 - Allows us to iterate over the values
- Common to iterate over the keys and access the values if you need both

Memory Diagram


```

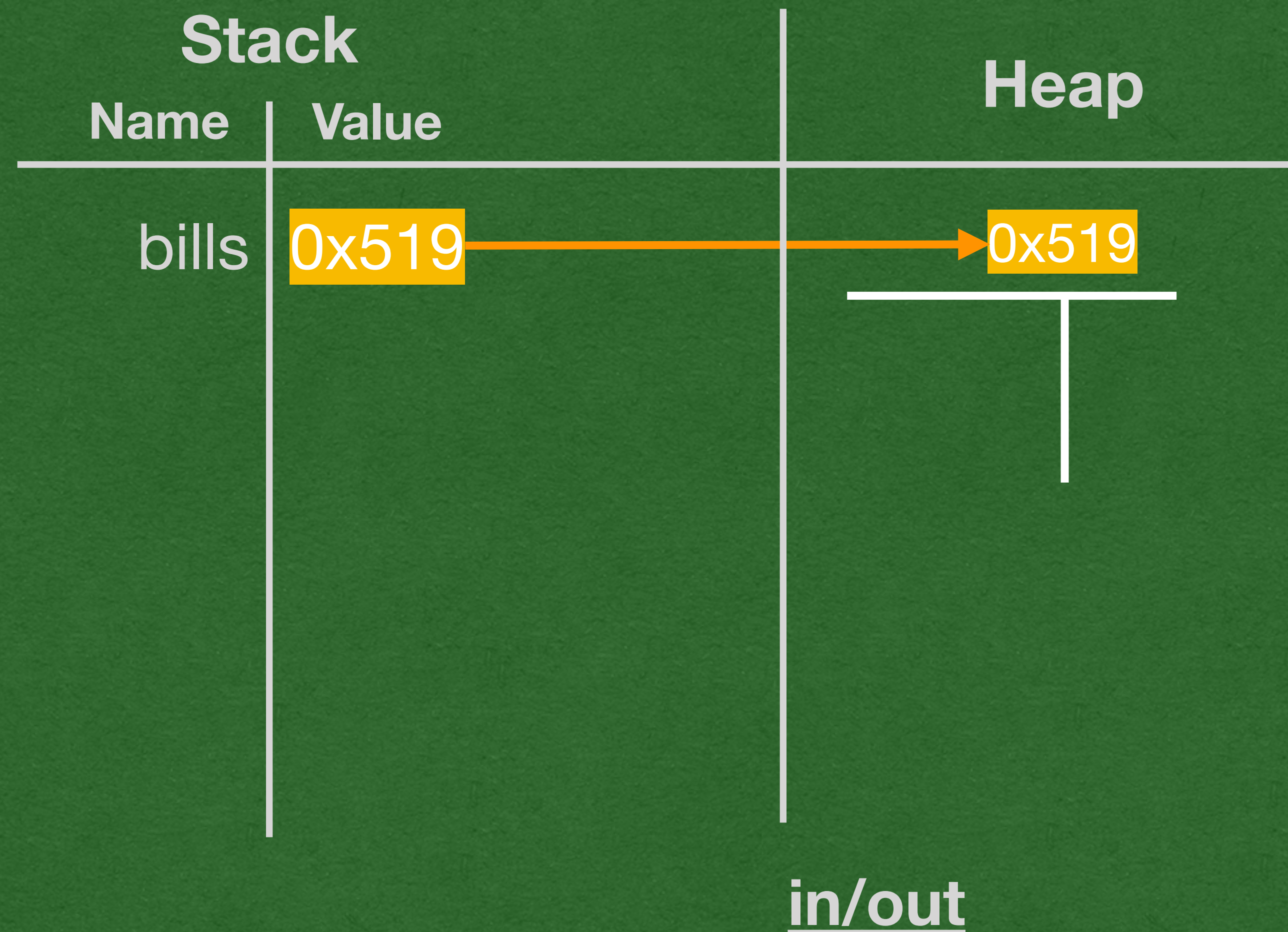
package week2;

import java.util.HashMap;

public class HashMap1 {
    ➡ public static void main(String[] args) {
        HashMap<String, Integer> bills = new HashMap<>();

        bills.put("Allen", 17);
        bills.put("Diggs", 14);
        System.out.print("What is Allen's number? ");
        System.out.println(bills.get("Allen"));
        System.out.println(bills);
        for (String key : bills.keySet()) {
            System.out.println(key);
        }
        for (Integer value : bills.values()) {
            System.out.println(value);
        }
        for (String key : bills.keySet()) {
            int value = bills.get(key);
            System.out.print(key + "'s number is: ");
            System.out.println(value);
        }
    }
}

```



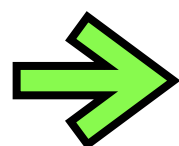
- HashMaps go in the heap
- Only a reference to the HashMap is stored on the stack


```
package week2;

import java.util.HashMap;

public class HashMap1 {
    public static void main(String[] args) {
        HashMap<String, Integer> bills = new HashMap<>();

        bills.put("Allen", 17);
        bills.put("Diggs", 14);
        System.out.print("What is Allen's number? ");
        System.out.println(bills.get("Allen"));
        System.out.println(bills);
        for (String key : bills.keySet()) {
            System.out.println(key);
        }
        for (Integer value : bills.values()) {
            System.out.println(value);
        }
        for (String key : bills.keySet()) {
            int value = bills.get(key);
            System.out.print(key + "'s number is: ");
            System.out.println(value);
        }
    }
}
```



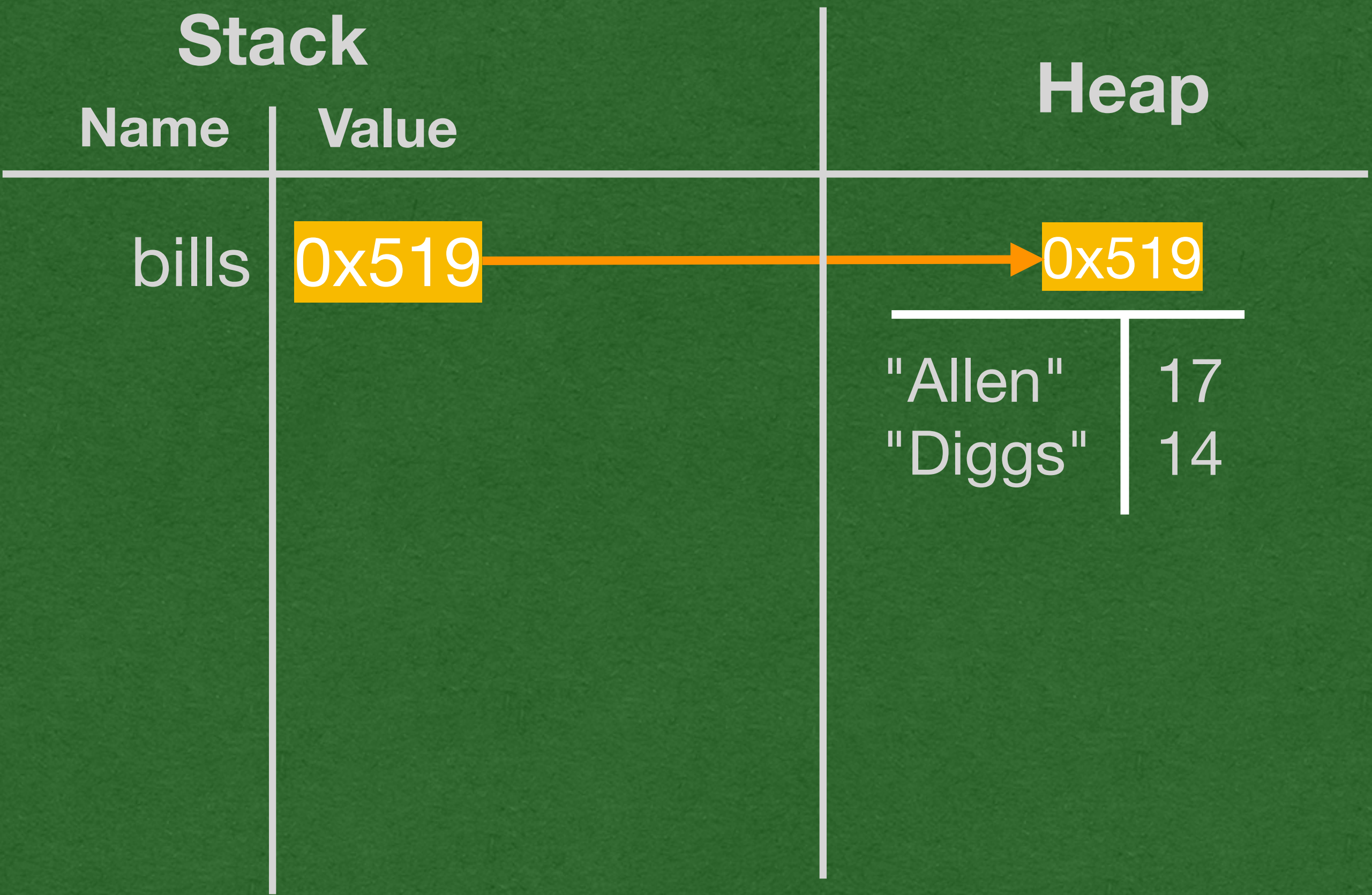
- HashMaps have columns for keys and values


```
package week2;

import java.util.HashMap;

public class HashMap1 {
    public static void main(String[] args) {
        HashMap<String, Integer> bills = new HashMap<>();

        bills.put("Allen", 17);
        bills.put("Diggs", 14);
        System.out.print("What is Allen's number? ");
        System.out.println(bills.get("Allen"));
        ➡ System.out.println(bills);
        for (String key : bills.keySet()) {
            System.out.println(key);
        }
        for (Integer value : bills.values()) {
            System.out.println(value);
        }
        for (String key : bills.keySet()) {
            int value = bills.get(key);
            System.out.print(key + "'s number is: ");
            System.out.println(value);
        }
    }
}
```



in/out

What is Allen's number? 17
{Diggs=14, Allen=17}

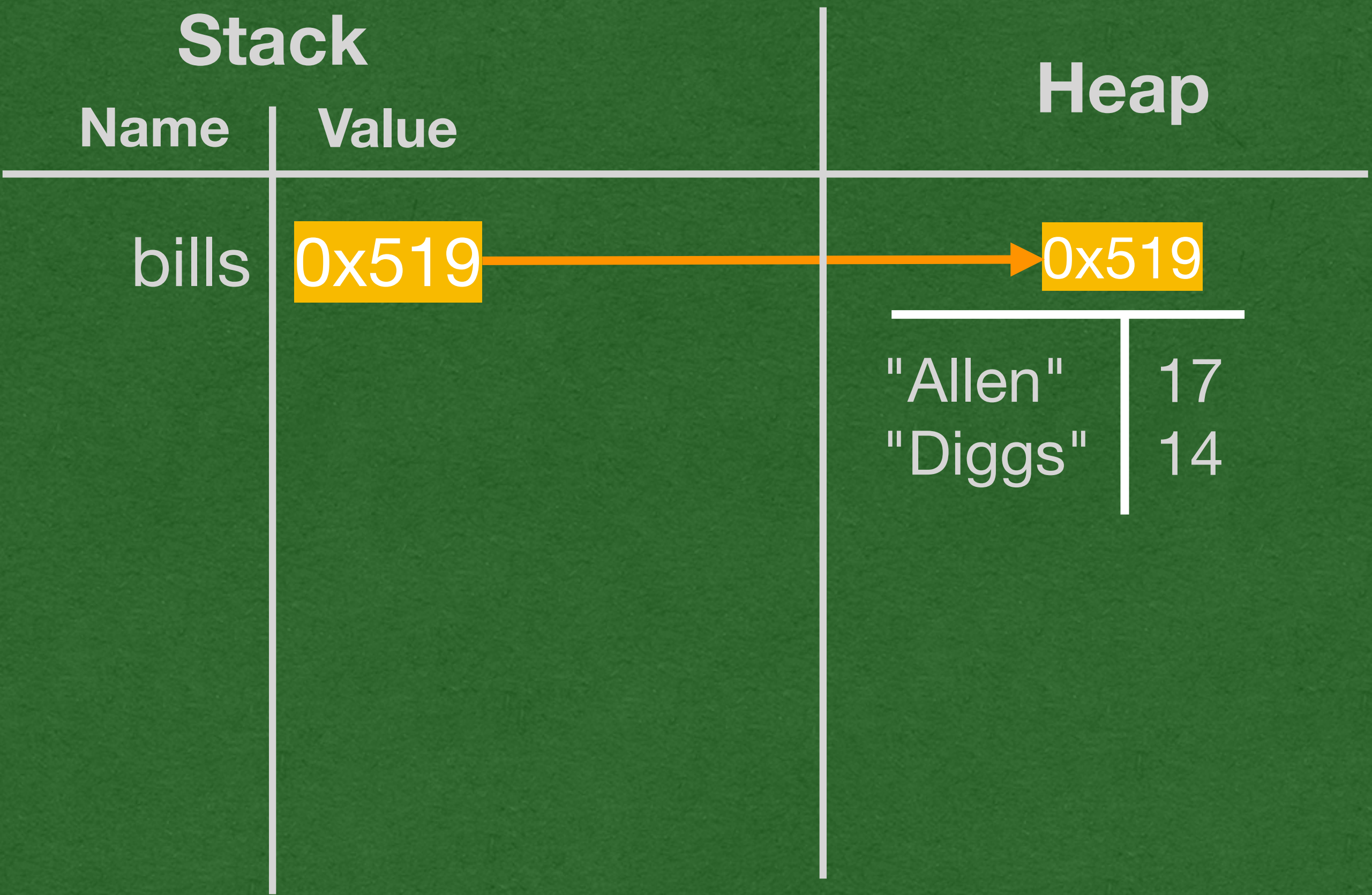
- HashMap prints as a list of key-value pairs in { } separated by commas
- Equal sign = separates each key from it's value


```
package week2;

import java.util.HashMap;

public class HashMap1 {
    public static void main(String[] args) {
        HashMap<String, Integer> bills = new HashMap<>();

        bills.put("Allen", 17);
        bills.put("Diggs", 14);
        System.out.print("What is Allen's number? ");
        System.out.println(bills.get("Allen"));
        ➡ System.out.println(bills);
        for (String key : bills.keySet()) {
            System.out.println(key);
        }
        for (Integer value : bills.values()) {
            System.out.println(value);
        }
        for (String key : bills.keySet()) {
            int value = bills.get(key);
            System.out.print(key + "'s number is: ");
            System.out.println(value);
        }
    }
}
```



in/out

What is Allen's number? 17
{Diggs=14, Allen=17}

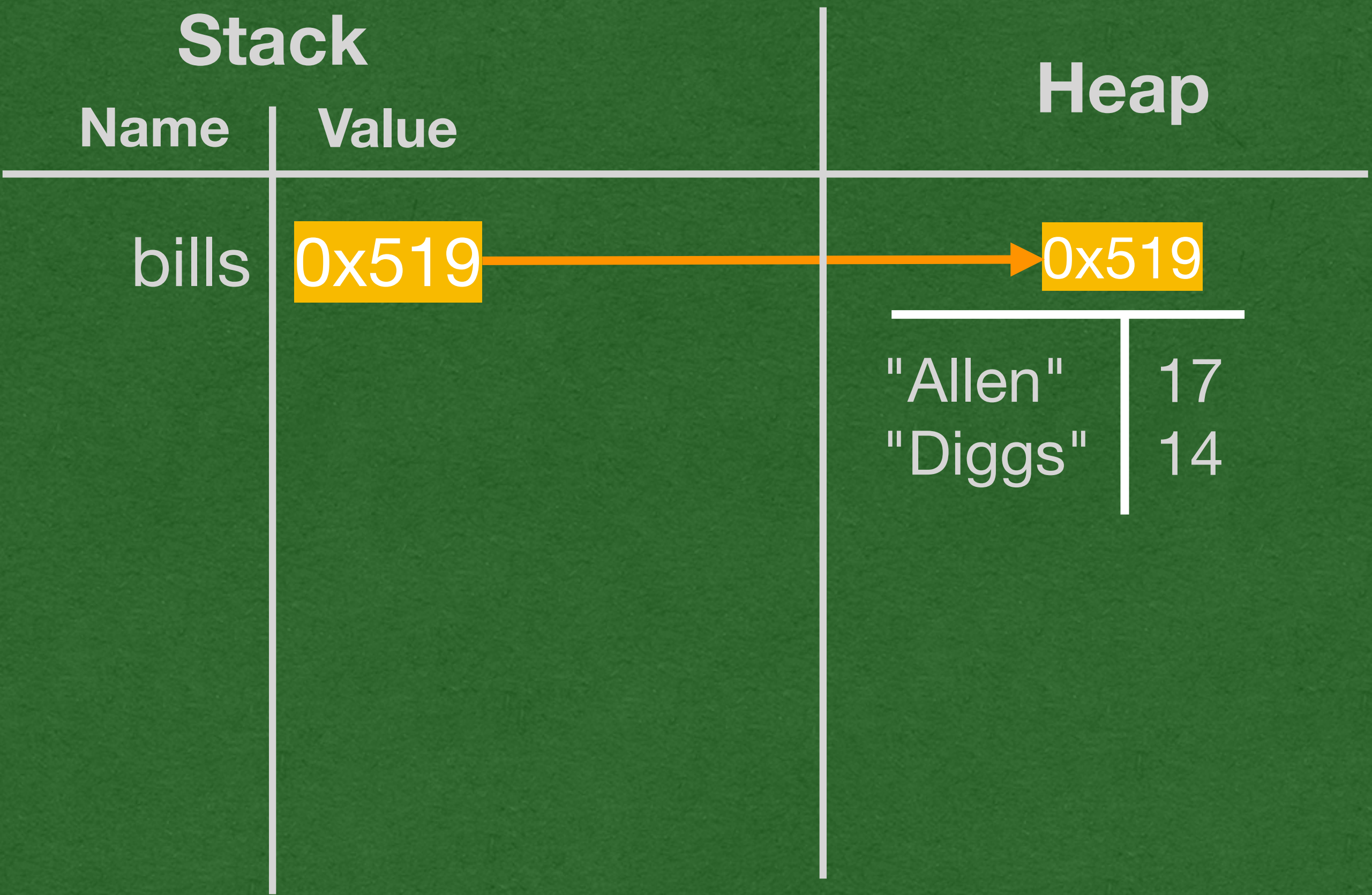
- Order does not matter in a HashMap!
- Notice how "Diggs" was printed before "Allen"
- No simple way to predict the order


```
package week2;

import java.util.HashMap;

public class HashMap1 {
    public static void main(String[] args) {
        HashMap<String, Integer> bills = new HashMap<>();

        bills.put("Allen", 17);
        bills.put("Diggs", 14);
        System.out.print("What is Allen's number? ");
        System.out.println(bills.get("Allen"));
        ➡ System.out.println(bills);
        for (String key : bills.keySet()) {
            System.out.println(key);
        }
        for (Integer value : bills.values()) {
            System.out.println(value);
        }
        for (String key : bills.keySet()) {
            int value = bills.get(key);
            System.out.print(key + "'s number is: ");
            System.out.println(value);
        }
    }
}
```



in/out
What is Allen's number? 17
{Diggs=14, Allen=17}

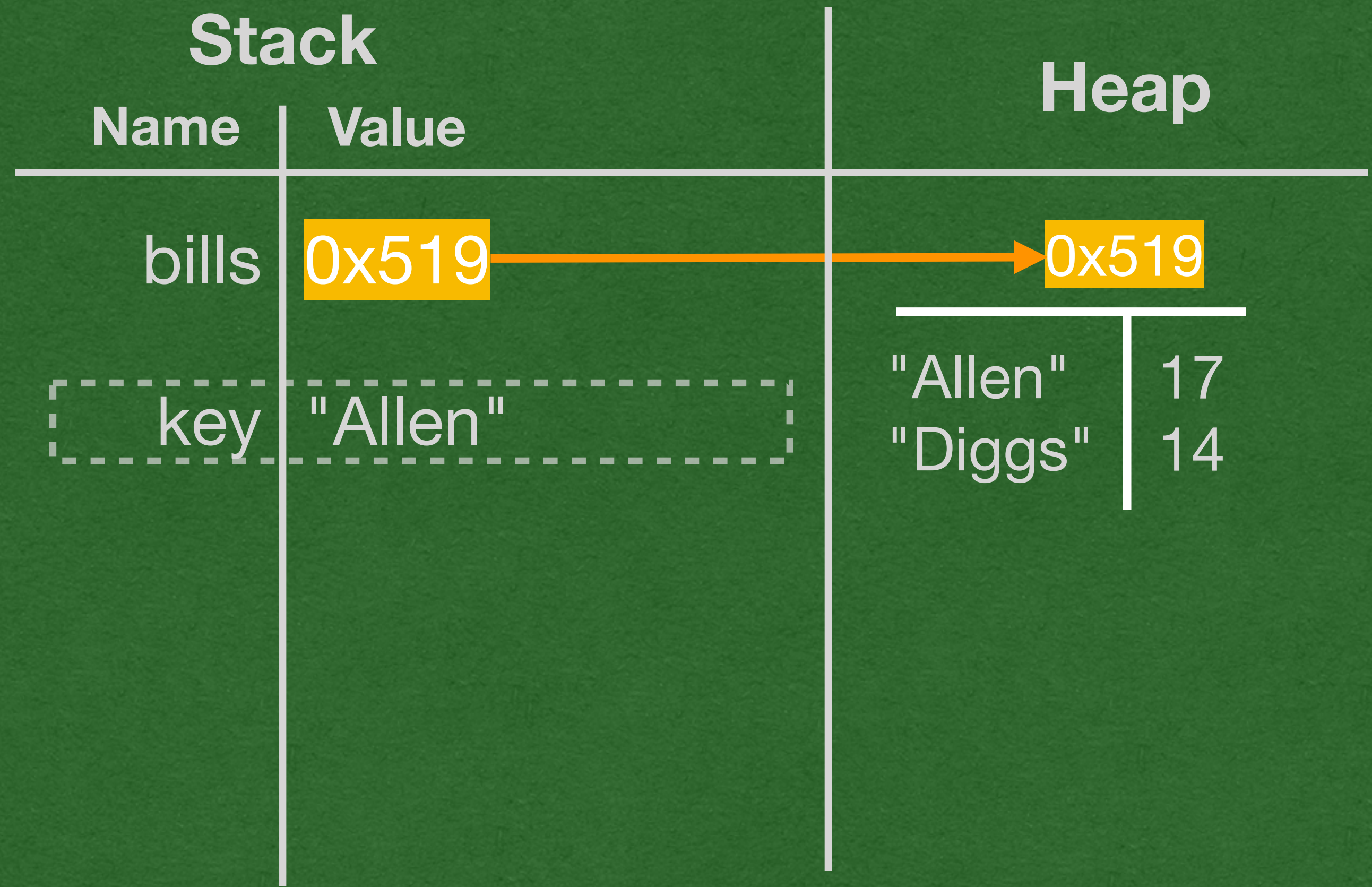
- In your memory diagrams, any order is acceptable for credit


```
package week2;

import java.util.HashMap;

public class HashMap1 {
    public static void main(String[] args) {
        HashMap<String, Integer> bills = new HashMap<>();

        bills.put("Allen", 17);
        bills.put("Diggs", 14);
        System.out.print("What is Allen's number? ");
        System.out.println(bills.get("Allen"));
        System.out.println(bills);
        for (String key : bills.keySet()) {
            System.out.println(key);
        }
        for (Integer value : bills.values()) {
            System.out.println(value);
        }
        for (String key : bills.keySet()) {
            int value = bills.get(key);
            System.out.print(key + "'s number is: ");
            System.out.println(value);
        }
    }
}
```



in/out

What is Allen's number? 17
{Diggs=14, Allen=17}
Allen

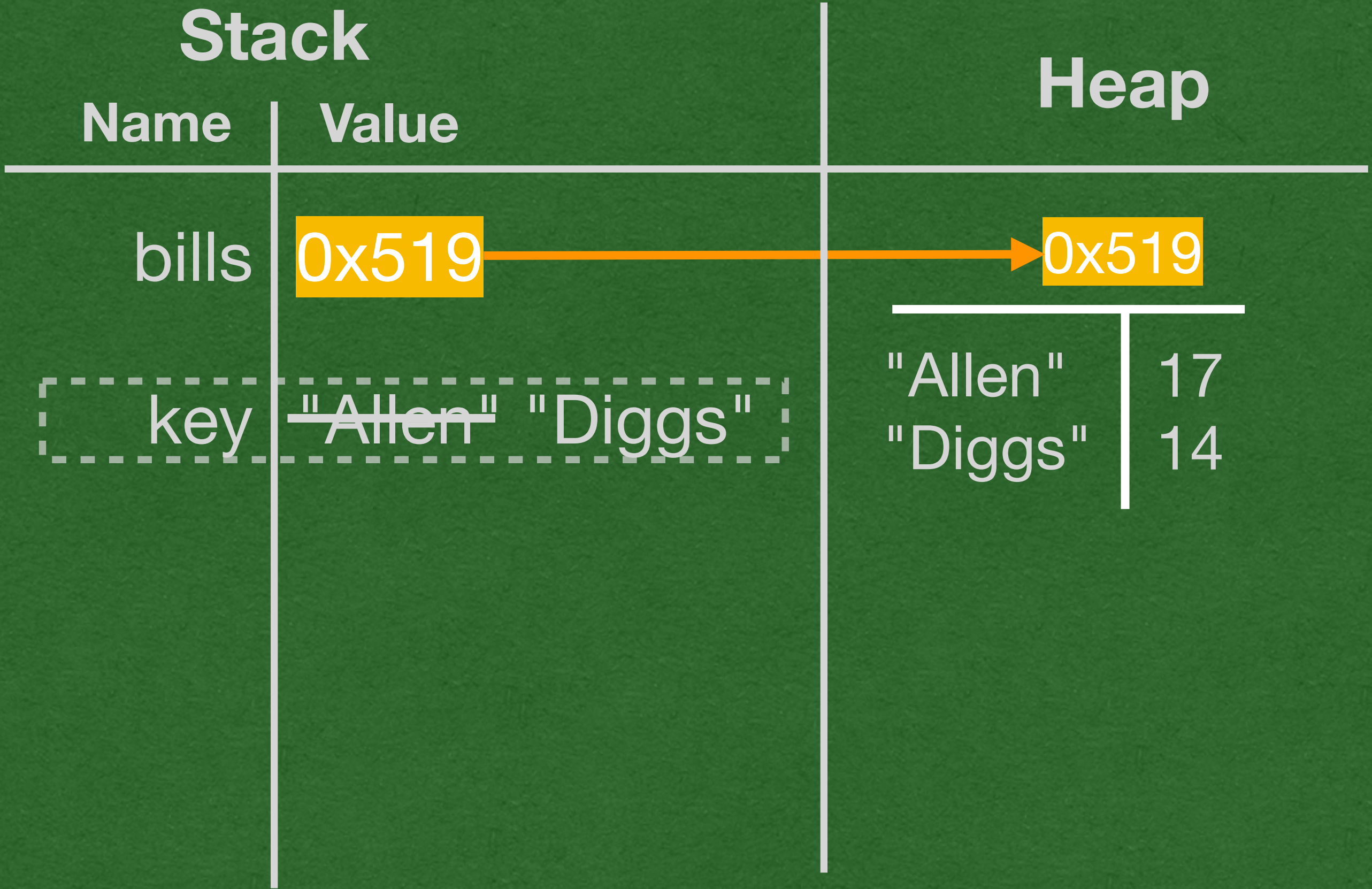
- Iterating over the keySet stores each key in the "key" variable and runs the body of the loop for each key


```
package week2;

import java.util.HashMap;

public class HashMap1 {
    public static void main(String[] args) {
        HashMap<String, Integer> bills = new HashMap<>();

        bills.put("Allen", 17);
        bills.put("Diggs", 14);
        System.out.print("What is Allen's number? ");
        System.out.println(bills.get("Allen"));
        System.out.println(bills);
        for (String key : bills.keySet()) {
            System.out.println(key);
        }
        for (Integer value : bills.values()) {
            System.out.println(value);
        }
        for (String key : bills.keySet()) {
            int value = bills.get(key);
            System.out.print(key + "'s number is: ");
            System.out.println(value);
        }
    }
}
```



in/out

What is Allen's number? 17
{Diggs=14, Allen=17}
Allen
Diggs

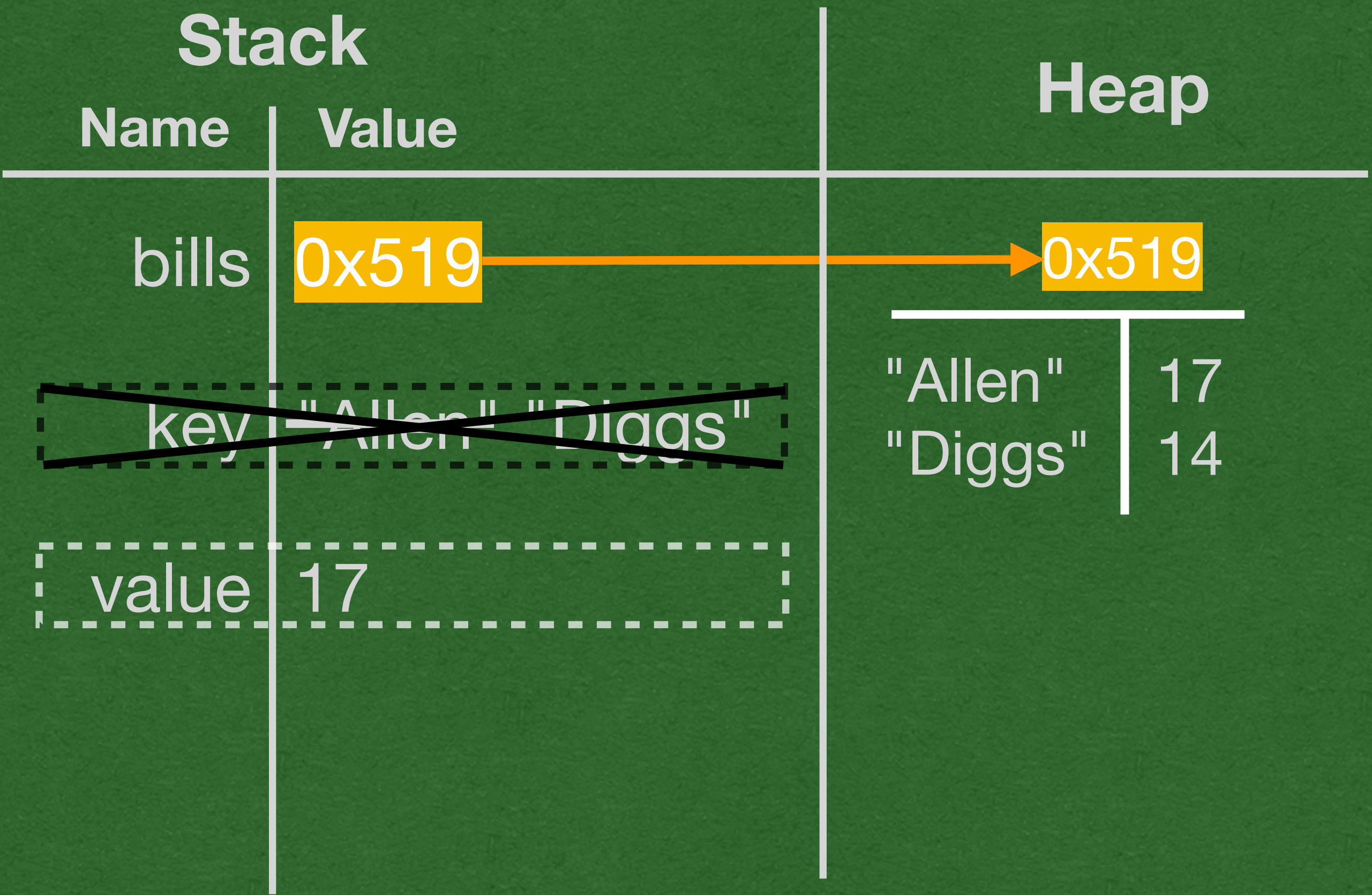
- Once we iterate over all the keys, the loops ends
- Note: If there are no key-value pairs in the HashMap, the loop body will never execute


```
package week2;

import java.util.HashMap;

public class HashMap1 {
    public static void main(String[] args) {
        HashMap<String, Integer> bills = new HashMap<>();

        bills.put("Allen", 17);
        bills.put("Diggs", 14);
        System.out.print("What is Allen's number? ");
        System.out.println(bills.get("Allen"));
        System.out.println(bills);
        for (String key : bills.keySet()) {
            System.out.println(key);
        }
        ➡ for (Integer value : bills.values()) {
            System.out.println(value);
        }
        for (String key : bills.keySet()) {
            int value = bills.get(key);
            System.out.print(key + "'s number is: ");
            System.out.println(value);
        }
    }
}
```



in/out

What is Allen's number? 17
{Diggs=14, Allen=17}
Allen
Diggs
17

- Iterating over the values only stores the values in the iteration variable


```
package week2;

import java.util.HashMap;

public class HashMap1 {
    public static void main(String[] args) {
        HashMap<String, Integer> bills = new HashMap<>();

        bills.put("Allen", 17);
        bills.put("Diggs", 14);
        System.out.print("What is Allen's number? ");
        System.out.println(bills.get("Allen"));
        System.out.println(bills);
        for (String key : bills.keySet()) {
            System.out.println(key);
        }
        for (Integer value : bills.values()) {
            System.out.println(value);
        }
        for (String key : bills.keySet()) {
            int value = bills.get(key);
            System.out.print(key + "'s number is: ");
            System.out.println(value);
        }
    }
}
```



in/out

What is Allen's number? 17
{Diggs=14, Allen=17}
Allen
Diggs
17
14

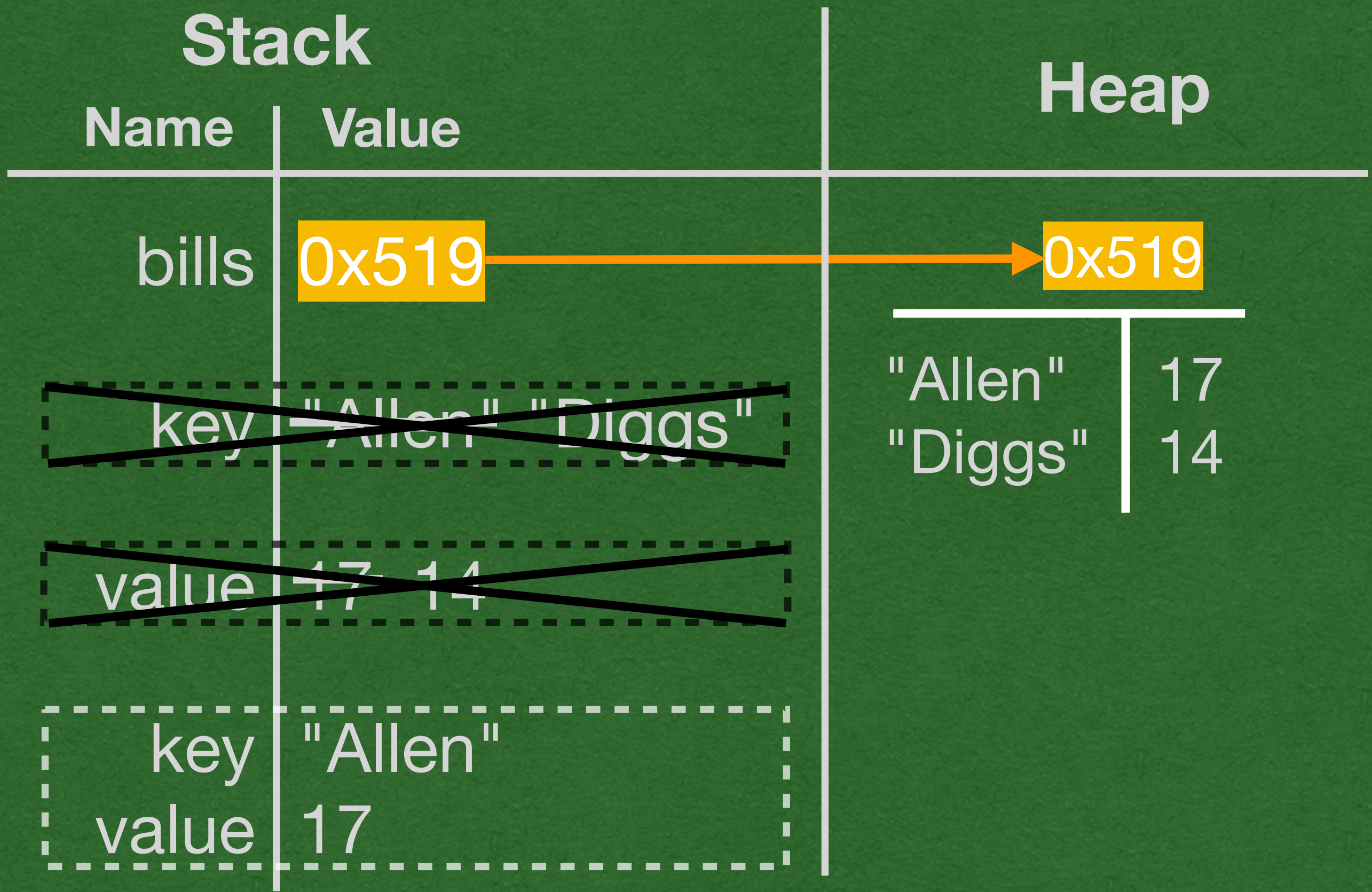
- Iterate until we run out of values


```
package week2;

import java.util.HashMap;

public class HashMap1 {
    public static void main(String[] args) {
        HashMap<String, Integer> bills = new HashMap<>();

        bills.put("Allen", 17);
        bills.put("Diggs", 14);
        System.out.print("What is Allen's number? ");
        System.out.println(bills.get("Allen"));
        System.out.println(bills);
        for (String key : bills.keySet()) {
            System.out.println(key);
        }
        for (Integer value : bills.values()) {
            System.out.println(value);
        }
        for (String key : bills.keySet()) {
            int value = bills.get(key);
            System.out.print(key + "'s number is: ");
            System.out.println(value);
        }
    }
}
```



in/out

What is Allen's number? 17
{Diggs=14, Allen=17}
Allen
Diggs
17
14
Allen's number is: 17

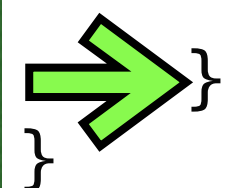
- If we iterate over the keys and get the values, we can access the key-value pairs


```
package week2;

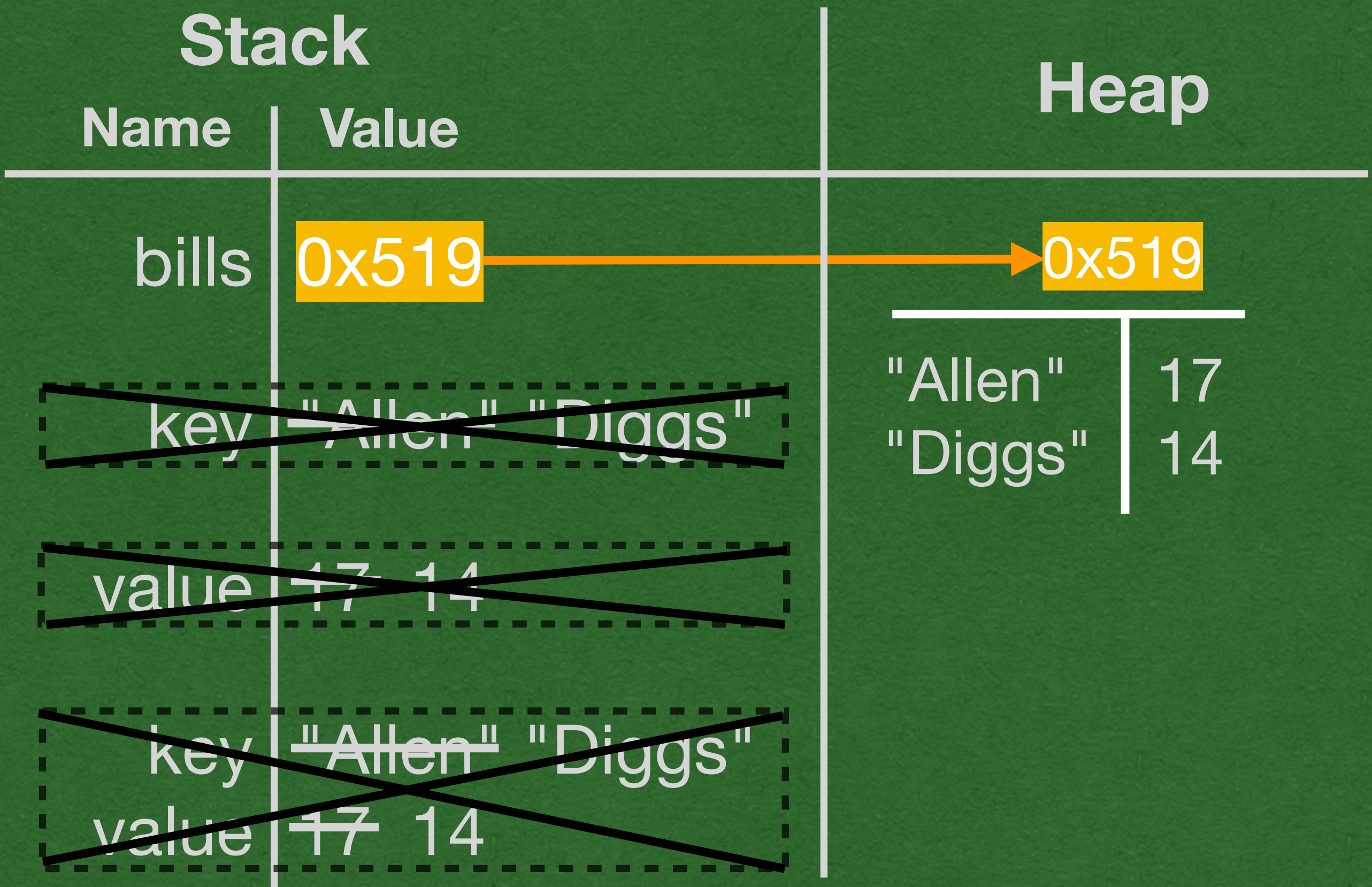
import java.util.HashMap;

public class HashMap1 {
    public static void main(String[] args) {
        HashMap<String, Integer> bills = new HashMap<>();

        bills.put("Allen", 17);
        bills.put("Diggs", 14);
        System.out.print("What is Allen's number? ");
        System.out.println(bills.get("Allen"));
        System.out.println(bills);
        for (String key : bills.keySet()) {
            System.out.println(key);
        }
        for (Integer value : bills.values()) {
            System.out.println(value);
        }
        for (String key : bills.keySet()) {
            int value = bills.get(key);
            System.out.print(key + "'s number is: ");
            System.out.println(value);
        }
    }
}
```



- Reach the end of main
- That's the end of the program



in/out

What is Allen's number? 17
{Diggs=14, Allen=17}
Allen
Diggs
17
14
Allen's number is: 17
Diggs's number is: 14