# CSE 143
# Lecture 2

**`ArrayIntList`**, binary search

slides created by Ethan Apter and Marty Stepp
http://www.cs.washington.edu/143/

## Lists

- Lists can be found everywhere
  - Shopping list
  - Top 10 list
  - List of students
  - List of items Mario has collected
  - List of types of lists (like this one)

- To keep things simple, we'll build a list of integers.
  - What is needed?
    - Something to keep track of the contents of the list (array)
    - Something to keep track of how many things are in the list (int)

# Lists

- We could write something like this:

```
public static void main(String[] args) {
   int[] list = new int[10];
   int size = 0;

   // add 10, -4, 23 to list
   list[0] = 10;
   list[1] = -4;
   list[2] = 23;
   size = 3;

   for (int i = 0; i < size; i++) {
      System.out.println("Element " + i + " is: " + list[i]);
   }
}
```

- What is wrong with this list?
  - List code not reusable (every person who wants a list must create their own array and manage it manually)
  - List code prone to error (what if I set the size to 4 instead of 3?)

3

# Lists

- We're going to make the notion of a list into a class (the blueprint for list objects).
- Each list object will contain an array and the list size.
- Each list object will protect (encapsulate) the data so that you can't "break" the list (e.g., can't change the size arbitrarily)
- Each list object will have a nice, friendly interface.

4

## ArrayIntList Client Code

- First we need some client code to test our new list class:

```
public class ArrayIntListClient {
    public static void main(String[] args) {
        // construct ArrayIntList list1
        // construct ArrayIntList list2
        // add 6, 43, 97 to list1
        // add 72, -8 to list2
        // print list1
        // print list2
    }
}
```

**We need to make an `ArrayIntList` class to construct**

5

## ArrayIntList

- What variables should be in **`ArrayIntList`**?
  - an array, to store the **`int`**s
  - an **`int`**, to store the size
- The size variable allows us to distinguish between our list's capacity (the amount the array can hold) and our list's size (the amount of valid data in the array)
- Variable code

```
int[] elementData = new int[100];
int size = 0;
```

6

# ArrayIntList

- We don't want to force the client to make these two variables
- Instead, we'll encapsulate the variables in an object

```
public class ArrayIntList {
    int[] elementData = new int[100];
    int size = 0;
}
```



# ArrayIntList Client Code

- Now we can update our client code to construct **ArrayIntList**s:

```
public class ArrayIntListClient {
    public static void main(String[] args) {
        ArrayIntList list1 = new ArrayIntList();
        ArrayIntList list2 = new ArrayIntList();
        // add 6, 43, 97 to list1
        // add 72, -8 to list2
        // print list1
        // print list2
    }
}
```

**Updated**

**We need to add values to the ArrayIntList**

## `ArrayIntList` Client Code: Attempt

- We might try something like this:

```
public class ArrayIntListClient {
    public static void main(String[] args) {
        ArrayIntList list1 = new ArrayIntList();
        ArrayIntList list2 = new ArrayIntList();
        // add 6, 43, 97 to list1
        list1.elementData[0] = 6;
        list1.elementData[1] = 43;
        list1.elementData[2] = 97;
        list1.size = 3;
        // add 72, -8 to list2
        // print list1
        // print list2
    }
}
```

**It works, but it's not an object-oriented approach (no encapsulation!)**



## Implementing `add`

| *index* | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* |
|---|---|---|---|---|---|---|---|---|---|---|
| *value* | 3 | 8 | 9 | 7 | 5 | 12 | 0 | 0 | 0 | 0 |
| *size* | 6 | | | | | | | | | |

- `list.add(42);`

| *index* | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* |
|---|---|---|---|---|---|---|---|---|---|---|
| *value* | 3 | 8 | 9 | 7 | 5 | 12 | **42** | 0 | 0 | 0 |
| *size* | **7** | | | | | | | | | |

- To add to end of list, just store element at index `size` and increase size:

```
elementData[size] = value;
size++;
```

# Static and Instance Methods

- A static add method would look something like this:

```
public static void add(int value) {
      // defines a static method add
      …
}
```

- While an instance add method would look like this:

```
public void add(int value) {
      // defines an instance method add
      …
}
```

- Notice the lack of the word **`static`** in the instance method

11

# Static and Instance Methods

- Static method call:

```
add(13);      // call on static method add
```

- However, instance methods require dot notation

- Instance method calls:

```
list1.add(1);// call on list1's instance method add
list2.add(7);// call on list2's instance method add
```

- The variable before the dot (**`list1`** and **`list2`** above) is the implicit parameter
  - The `add` method accesses `elementData` and `size` belonging to the implicit parameter

12

# ArrayIntList

- So our updated **ArrayIntList**, containing an instance add method, looks like this:

```
public class ArrayIntList {
    int[] elementData = new int[100];
    int size = 0;

    public void add(int value) {
        elementData[size] = value;
        size++;
    }
}
```

13

# ArrayIntList Client Code

- Now we can update our client code to add values to the **ArrayIntLists**:

```
public class ArrayIntListClient {
    public static void main(String[] args) {
        ArrayIntList list1 = new ArrayIntList();
        ArrayIntList list2 = new ArrayIntList();
        list1.add(6);
        list1.add(43);
        list1.add(97);
        list2.add(72);
        list2.add(-8);

        // print list1
        // print list2
    }
}
```

**Updated**

**We still need to print our lists**

14

# But Does `add` Work?

- One quick way to check is to print just the sizes of the lists:

```
public class ArrayIntListClient {
        public static void main(String[] args) {
                ArrayIntList list1 = new ArrayIntList();
                ArrayIntList list2 = new ArrayIntList();
                list1.add(6);
                list1.add(43);
                list1.add(97);
                list2.add(72);
                list2.add(-8);
                System.out.println(list1.size);
                System.out.println(list2.size);
        }
}
```

Updated



# `print`

- This is the print method from yesterday's section:

```
public static void print(int[] list) {
        if (list.length == 0) {
                System.out.println("[]");
        } else {
                System.out.print("[" + list[0]);
                for (int i = 1; i < list.length; i++) {
                        System.out.print(", " + list[i]);
                }
                System.out.println("]");
        }
}
```

# print

- Let's update **print** to be an instance method:

```
public void print() {
    if (size == 0) {
        System.out.println("[]");
    } else {
        System.out.print("[" + elementData[0]);
        for (int i = 1; i < size; i++) {
            System.out.print(", " + elementData[i]);
        }
        System.out.println("]");
    }
}
```

- Revisions: removed **static**, removed parameter, changed **list** to **elementData**, changed **list.length** to **size**

17

# ArrayIntList Client Code

- Now we can update our client code to use the print method:

```
public class ArrayIntListClient {
      public static void main(String[] args) {
            ArrayIntList list1 = new ArrayIntList();
            ArrayIntList list2 = new ArrayIntList();
            list1.add(6);
            list1.add(43);
            list1.add(97);
            list2.add(72);
            list2.add(-8);
            list1.print();
            list2.print();
      }
}
```

**Updated**

18

## `print`'s Limitations

- **`print`** forces us to always print to System.out
  - What if we want to send it to a file instead?
  - What if we're working on a Graphical User Interface (GUI) and we want the output to appear on a specific part of the screen?
- **`print`** forces us to print only a single **`ArrayIntList`** per line
  - What if we want two or more on the same line?
- If the client wants to do either of the above, she will have to handle the details herself. That's not good design on our part.
- How can we fix this?

19

## `print`

- Let's update **`print`** to return a **`String`**:

```
public String print() {
    if (size == 0) {
        return "[]";
    } else {
        String result = "[" + elementData[0];
        for (int i = 1; i < size; i++) {
            result += ", " + elementData[i];
        }
        return result + "]";
    }
}
```

20

# `ArrayIntList` Client Code

- Now we can update our client code to use the updated print method:

```
public class ArrayIntListClient {
        public static void main(String[] args) {
                ArrayIntList list1 = new ArrayIntList();
                ArrayIntList list2 = new ArrayIntList();
                list1.add(6);
                list1.add(43);
                list1.add(97);
                list2.add(72);
                list2.add(-8);
                System.out.println(list1.print());
                System.out.println(list2.print());
        }
}
```

**Updated**

- But this doesn't seem like much of an improvement

21

# toString

- Let's rename print to toString:

```
public String toString() {
    if (size == 0) {
        return "[]";
    } else {
        String result = "[" + elementData[0];
        for (int i = 1; i < size; i++) {
            result += ", " + elementData[i];
        }
        return result + "]";
    }
}
```

- Java will automatically call `toString` when we pass our `ArrayIntList` to `System.out.println`

22

# `ArrayIntList` Client Code

- Now we can update our client code to use our toString method:

```
public class ArrayIntListClient {
        public static void main(String[] args) {
                ArrayIntList list1 = new ArrayIntList();
                ArrayIntList list2 = new ArrayIntList();
                list1.add(6);
                list1.add(43);
                list1.add(97);
                list2.add(72);
                list2.add(-8);
                System.out.println(list1);
                System.out.println(list2);
        }
}
```

**Updated**



# Implementing `add` (2)

- How do we add to the middle or end of the list?

| *index* | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* |
|---|---|---|---|---|---|---|---|---|---|---|
| *value* | 3 | 8 | 9 | 7 | 5 | 12 | 0 | 0 | 0 | 0 |
| *size* | 6 | | | | | | | | | |

– `list.add(3, 42);` `// insert 42 at index 3`

| *index* | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* |
|---|---|---|---|---|---|---|---|---|---|---|
| *value* | 3 | 8 | 9 | **42** | 7 | 5 | 12 | 0 | 0 | 0 |
| *size* | **7** | | | | | | | | | |

# Implementing `add` (2) cont.

- Adding to the middle or front is hard *(see book ch 7.4)*
  - must *shift* nearby elements to make room for the new value

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| value | 3 | 8 | 9 | 7 | 5 | 12 | 0 | 0 | 0 | 0 |
| size | 6 | | | | | | | | | |

  - `list.add(3, 42);` `// insert 42 at index 3`

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| value | 3 | 8 | 9 | **42** | **7** | **5** | **12** | 0 | 0 | 0 |
| size | **7** | | | | → | | | | | |

  - Note: The order in which you traverse the array matters!



# Implementing `add` (2) code

```
public void add(int index, int value) {
    for (int i = size; i > index; i--) {
        elementData[i] = elementData[i - 1];
    }
    elementData[index] = value;
}
```

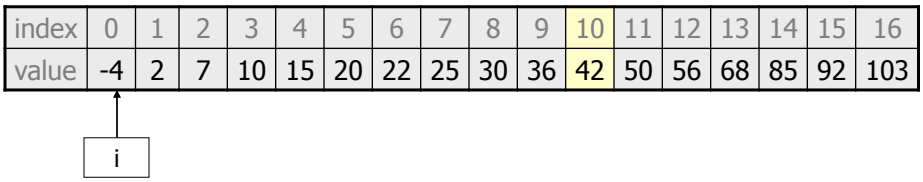
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| value | 3 | 8 | 9 | 7 | 5 | 12 | 0 | 0 | 0 | 0 |
| size | 6 | | | | | | | | | |

- `list.add(3, 42);`

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| value | 3 | 8 | 9 | **42** | **7** | **5** | **12** | 0 | 0 | 0 |
| size | **7** | | | | → | | | | | |

# Sequential search

- **sequential search**: Locates a target value in an array/list by examining each element from start to finish.
  - How many elements will it need to examine?
  - Example: Searching the array below for the value **42**:

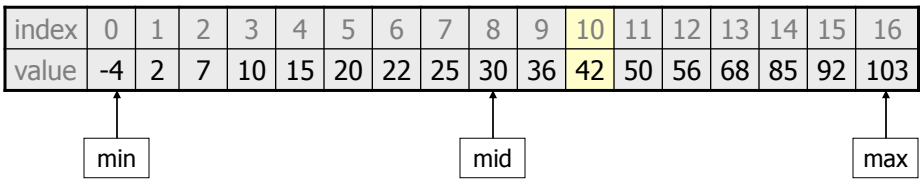



  - Notice that the array is sorted. Could we take advantage of this?



# Binary search (13.1)

- **binary search**: Locates a target value in a *sorted* array/list by successively eliminating half of the array from consideration.
  - How many elements will it need to examine?
  - Example: Searching the array below for the value **42**:



# The `Arrays` class

- Class `Arrays` in `java.util` has many useful array methods:

| Method name | Description |
|---|---|
| `binarySearch(`**array**`, `**value**`)` | returns the index of the given value in a *sorted* array (or < 0 if not found) |
| `binarySearch(`**array**`, `**minIndex**`, `**maxIndex**`, `**value**`)` | returns index of given value in a *sorted* array between indexes *min* / *max* - 1 (< 0 if not found) |
| `copyOf(`**array**`, `**length**`)` | returns a new resized copy of an array |
| `equals(`**array1**`, `**array2**`)` | returns `true` if the two arrays contain same elements in the same order |
| `fill(`**array**`, `**value**`)` | sets every element to the given value |
| `sort(`**array**`)` | arranges the elements into sorted order |
| `toString(`**array**`)` | returns a string representing the array, such as `"[10, 30, -25, 17]"` |

- Syntax: `Arrays.`**methodName**`(`**parameters**`)`

29

# `Arrays.binarySearch`

```
// searches an entire sorted array for a given value
// returns its index if found;  a negative number if not found
// Precondition: array is sorted
Arrays.binarySearch(array, value)

// searches given portion of a sorted array for a given value
// examines minIndex (inclusive) through maxIndex (exclusive)
// returns its index if found;  a negative number if not found
// Precondition: array is sorted
Arrays.binarySearch(array, minIndex, maxIndex, value)
```

- The `binarySearch` method in the `Arrays` class searches an array very efficiently if the array is sorted.
  - You can search the entire array, or just a range of indexes (useful for "unfilled" arrays such as the one in `ArrayIntList`)
  - If the array is not sorted, you may need to sort it first

30

# Using `binarySearch`

```
// index    0  1  2  3   4   5   6   7   8   9  10  11  12  13  14  15
int[] a = {-4, 2, 7, 9, 15, 19, 25, 28, 30, 36, 42, 50, 56, 68, 85, 92};
int index  = Arrays.binarySearch(a, 0, 16, 42);   // index1 is 10
int index2 = Arrays.binarySearch(a, 0, 16, 21);   // index2 is -7
```

- `binarySearch` returns the index where the value is found
- if the value is *not* found, `binarySearch` returns:

  `-(insertionPoint + 1)`

  - where `insertionPoint` is the index where the element *would* have been, if it had been in the array in sorted order.
  - To insert the value into the array, negate `insertionPoint` + 1

```
int indexToInsert21 = -(index2 + 1);  // 6
```

31