CSE 143
Lecture 2

`ArrayIntList`

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

`ArrayList` Review

- **`ArrayList`** is an **`Object`**
  - It is one of the lists within the Java Collections Framework

- **`ArrayList`**s can store any kind of **`Object`**
  - **`String`**, **`Integer`**, **`Scanner`**, etc
  - …but they can't store primitives like `int` and `double`

- **`ArrayList`**s can perform many useful list operations
  - **`add(value)  // adds given value to end of list`**
  - **`get(index)  // returns value at given index`**
  - **`size()  // returns size of list`**
  - etc

`ArrayList` Review

- Sample **`ArrayList`** code:

```
import java.util.*; // must occur before using ArrayList

// construct new ArrayList of Strings
ArrayList<String> list = new ArrayList<String>();

// add a few elements, and print
list.add("Apter");          // adds "Apter" to end
list.add(0, "Ethan");       // adds "Ethan" to index 0, shifting others
System.out.println(list);   // prints [Ethan, Apter]

// remove an element, and print
list.remove(0);             // removes "Ethan" and shifts others
System.out.println(list);   // prints [Apter]
```

- Read chapter 10.1 or Java API for more information

Learning from `ArrayList`

- We're going to build our own version of **`ArrayList`**
- However, the full version of **`ArrayList`** is complicated
  - it can store any kind of **`Object`**
  - it has more than 20 methods
- So we will make a simpler version called **`ArrayIntList`**
  - it will store only one kind of thing (**`int`**s)
  - it will have fewer methods
  - it will improve our understanding of arrays and **`Object`**s

`ArrayIntList` Client Code

- First we need some client code:

```
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

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;
```

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

ArrayIntList

- Instead, we want to write an add method
- The code within the add method will look like this:

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

- The parameters for the add method will be:

```
value
```

- But what about **`elementData`** and **`size`**?
  - Our add method will be an *instance method* instead of a static method
  - It will have an *implicit parameter* containing **`elementData`** and **`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

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

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++;
      }
}
```

`ArrayIntList` Client Code

- Now we can update our client code to add values to the **`ArrayIntList`s**:

```
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

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

- Fortunately, we can do even better with jGRASP

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**

`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

`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?

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 + "]";
    }
}
```

`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

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`**

`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