Lecture 13: ArrayLists

AP Computer Science Principles

Problem with Arrays

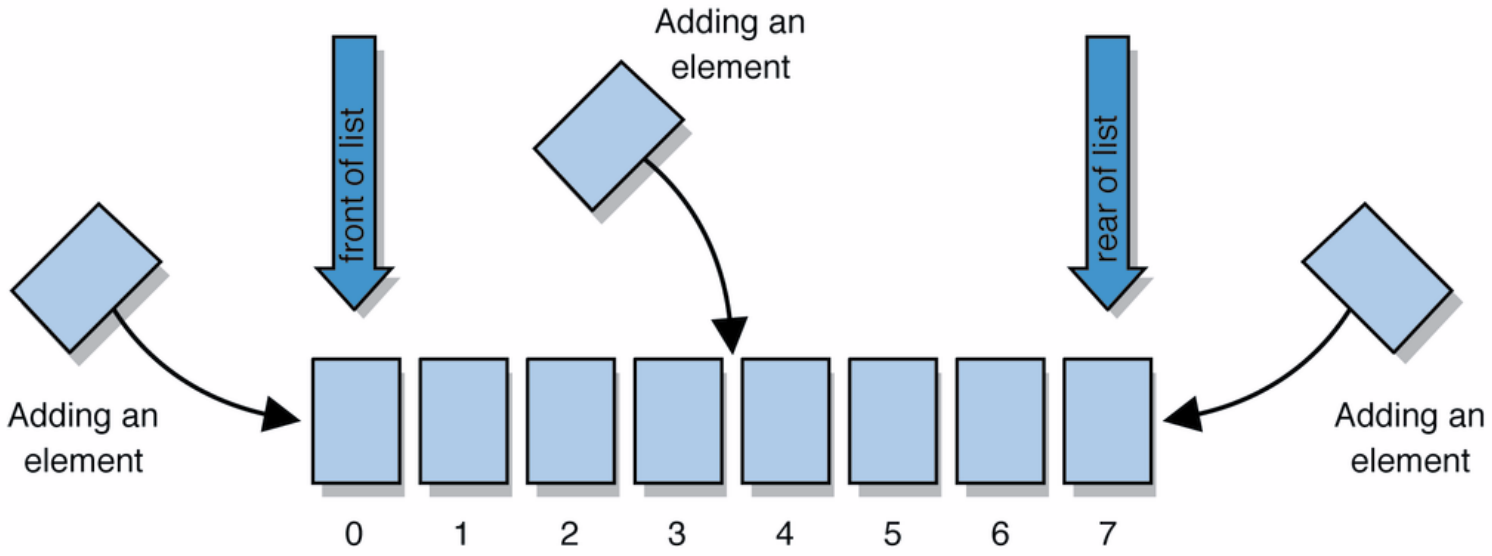
Ask the user to enter a list of words, save the words in an array.

```
String[] allWords = new String[1000];
int wordCount = 0;
```

- Problem: You don't know how many words the user will enter.
  - Hard to create an array of the appropriate size.
  - Can not lengthen an array by adding elements to it.
  - Elements of arrays cannot be removed.
- Luckily, there are other ways to store data besides in an array.

Lists

- **list**: a collection storing an ordered sequence of elements
  - each element is accessible by a 0-based **index**
  - a list has a **size** (number of elements that have been added)
  - elements can be added to the front, back, or elsewhere
  - in Java, a list can be represented as an **`ArrayList`** object



Idea of a arraylist

- Rather than creating an array of boxes, create an object that represents a "list" of items. (initially an empty list.)

```
[]
```

- You can add items to the list.
  - The default behavior is to add to the end of the list.

```
[hello, ABC, goodbye, okay]
```

- The list object keeps track of the element values that have been added to it, their order, indexes, and its total size.
  - Think of an "array list" as an automatically resizing array object.

`ArrayList` methods (10.1)

`add(`**value**`)`	appends value at end of list
`add(`**index**`, `**value**`)`	inserts given value just before the given index, shifting subsequent values to the right
`E get(`**index**`)`	returns the value at given index
`E remove(`**index**`)`	**removes/returns** value at given index, shifting subsequent values to the left
`E set(`**index**`, `**value**`)`	replaces value at given index with given value, **returns** element that was previously at index.
`int size()`	returns the number of elements in list

Note: E refers to the type that is being returned. This depends on the type of objects in the Arraylist.

Type Parameters (Generics)

```
ArrayList<Type> name = new ArrayList<Type>();
```

- When constructing an `ArrayList`, you must specify the type of elements it will contain between < and >.
  - Allows the same `ArrayList` class to store lists of different types.

```
ArrayList<String> names = new ArrayList<String>();
names.add("Marty Stepp");
names.add("Stuart Reges");

//notice the difference between arraylist & array
String[] names= new String[10];
```

`ArrayList` vs. array

- construction

```
String[] names = new String[5];
ArrayList<String> list = new ArrayList<String>();
```

- storing a value

```
names[0] = "Jessica";
list.add("Jessica");
```

- retrieving a value

```
String s = names[0];
String s = list.get(0);
```

ArrayList vs. array

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

list.add("Michael");
list.add("Jessica");
list.add("Lee");  //{"Michael", "Jessica", "Lee"}
list.add(1,"Sarah");//{"Michael", "Sarah","Jessica", "Lee"}
String store = list.set(2,"Mary")
//{Michael,Sarah,Mary,Lee}, store="Jessica"

String store2 = list.get(3);//store2="Lee"

String store3 = list.remove(1);
//{Michael,Mary,Lee}, store3="Sarah"
```

`ArrayList` vs. array 2

- doing something to each value that starts with `"B"`

```
for (int i = 0; i < names.length; i++) {
    if (names[i].equals(“John”) {
        // do something...
    }
}
```

```
for (int i = 0; i < list.size(); i++) {
    if (list.get(i).equals(“John”)) {
        // do something...
    }
}
```

`ArrayList` of primitives?

- The type you specify when creating an `ArrayList` **must be an object type**; **it cannot be a primitive type**.

```
// illegal -- int cannot be a type parameter
ArrayList<int> list = new ArrayList<int>();
```

- But we can still use `ArrayList` with primitive types by using *wrapper* classes in their place.

```
// creates a list of ints
ArrayList<Integer> list = new ArrayList<Integer>();
```

- Once you construct the list, use it with primitives as normal:

```
ArrayList<Float> grades = new ArrayList<Float>();
grades.add(3.2); //autoboxing
grades.add(2.7);
float myGrade = grades.get(0);//auto-unboxing
```

Wrapper Classes

Primitive Type	Wrapper Type
`int`	`Integer`
`float`	`Float`
`boolean`	`Boolean`

Java allows wrapper object for each of its primitive types.

Wrapper Classes

Integer and Float are wrapper classes…not Rapper Classes.

These are Rapper Classes:

```
class Tupac{…}
class Biggie{…}
class JayZ{…}
Class KendrickLamar{…}
```

ArrayList "mystery"

```
ArrayList<Integer> list = new ArrayList<Integer>();
for (int i = 1; i <= 10; i++) {
    list.add(10 * i);
}
```

- What is in the arraylist list?

```
[10, 20, 30, 40, ..., 100]
```

ArrayList "mystery"

```
ArrayList<Integer> list = new ArrayList<Integer>();
for (int i = 1; i <= 10; i++) {
    list.add(10 * i);// [10, 20, 30, 40, ..., 100]
}
```

- What is the output of the following code?

```
for (int i = 0; i < list.size(); i++) {
    list.remove(i);
}
println(list);
```

- Answer:

```
[20, 40, 60, 80, 100]
```

Remove elements

```
// Removes all elements with even values from the given
   list.
void filterEvens(ArrayList<Integer> list) {
    for (int i = 0; i <list.size(); i++) {
        int n = list.get(i);
        if (n % 2 == 0) {
            list.remove(i);
        }
    }
}
// INCORRECT!!
```

Solution 1

```
// Removes all elements with even values from the given
// list.
void filterEvens(ArrayList<Integer> list) {
    for (int i = 0; i <list.size(); i++) {
        int n = list.get(i);
        if (n % 2 == 0) {
            list.remove(i);
            i--; // prevents skipping
        }
    }
}

// CORRECT!!
```

Solution 2

Or alternatively, we can traverse the array backwards; this eliminates the need to do i--.

```
void filterEvens(ArrayList<Integer> list) {
    for (int i = list.size() - 1; i >= 0; i--)
  {
        int n = list.get(i);
        if (n % 2 == 0) {
            list.remove(i);
        }
    }
}
// Correct Version, go backwards, no need to do
// i--.
```

Out-of-bounds

- Legal indexes are between **0** and the **list's size() - 1**.
  - Reading or writing any index outside this range will cause an `IndexOutOfBoundsException`.

```
ArrayList<String> names = new ArrayList<String>();
names.add("Marty");    names.add("Kevin");
names.add("Vicki");    names.add("Larry");
println(names.get(0));          // okay
println(names.get(3));          // okay
println(names.get(-1));         // exception
names.add(9, "Aimee");                    // exception
```

index	*0*	*1*	*2*	*3*
value	Marty	Kevin	Vicki	Larry

ArrayList "mystery" 2

```
ArrayList<Integer> list = new ArrayList<Integer>();
for (int i = 1; i <= 5; i++) {
    list.add(2 * i);
}
// [2, 4, 6, 8, 10]
```

- What is the output of the following code?

```
int size = list.size(); // size = 5
for (int i = 0; i < size; i++) {
    list.add(i, 42);   // add 42 at index i
}
System.out.println(list);
```

- Answer:

```
[42, 42, 42, 42, 42, 2, 4, 6, 8, 10]
```

ArrayList "mystery" 2

```
ArrayList<Integer> list = new ArrayList<Integer>();
for (int i = 1; i <= 5; i++) {
    list.add(2 * i);   // [2, 4, 6, 8, 10]
}
```

- What is the output of the following code?

```
for (int i = 0; i < list.size(); i++) {
    list.add(i, 42);   // add 42 at index i
}
println(list);
```

- Answer: Infinite Loop!

ArrayList as parameter

```
void name(ArrayList<Type> name) {
```

- Example:

```
// Removes all plural words from the given list.
void removePlural(ArrayList<String> list) {
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i).endsWith("s")) {
                list.remove(i);
                i--;
            }
        }
    }
```

ArrayList as parameter

- You can also return a list:

```
ArrayList<Type>   name(ArrayList<Type> name) {
```

- Example:

```
// add all plural words from the given list to a
// new arraylist and return the new arraylist
ArrayList<String> addPlural(ArrayList<String> list) {
    ArrayList<String> result = new ArrayList<String>();

    for (int i = 0; i < list.size(); i++) {
            if (list.get(i).endsWith("s")) {
                result.add(list.get(i));
            }
        }
    return result;
    }
```

Lab 1

Download the files ArrayListLab.pde and Ball.pde on classroom.

Follow the directions in the comments to complete the lab.

This lab allows you to create an arraylist of Ball objects and then remove them as they touch the mouse.

Lab 2

Download the ShootLab from the website.

Follow the directions in the comments to complete the lab.

This lab allows you to “shoot” ball objects and remove them.

This can be modified to some version of the asteroid shooting game.

References

Part of this lecture is taken from the following book.

1) Stuart Reges and Marty Stepp. Building Java Programs: A Back to Basics Approach. Pearson Education. 2008.