Arrays & Array Lists

MSCI 240: Algorithms & Data Structures

slides by Mark Hancock

lecture summary

space in memory

arrays

`ArrayList`

`ArrayList` vs. arrays

generics

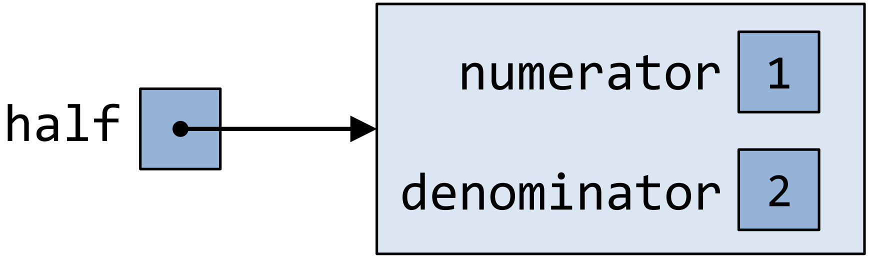
space in memory

primitive types	bytes	default value
byte	1	0
short	2	0
int	4	0
long	8	0
float	4	0.0f
double	8	0.0
boolean	n/a	false
char	2	'\u0000'

how much space does `Fraction` take?

```
Fraction half = new Fraction(1,2);
```

```
   4 bytes for numerator
+  4 bytes for denominator
+  4 bytes for reference
  12 bytes total
```



slides by Mark Hancock

space in memory summary

each primitive type has a fixed size

size of an object is the total of the size of each field

a reference to an object (i.e., the variable pointing to it) also takes up space in memory (4 bytes)

arrays

a data structure built in to Java with:

- contiguous storage of single data type
- compact space requirements
- random access in “constant time”
- fixed length

contiguous storage of single data type

- each element of the array is stored adjacent to the previous and next element in memory

compact space requirements

- size of elements (# elements × size of data type)
- size of length information (4 bytes)
- size of reference to array (4 bytes)

random access in “constant time”

let t_i be time to access i^{th} element

$$\therefore t_0 = t_1 = t_2 = \cdots = t_{length-1}$$

fixed length

once array is initialized, can’t change it’s size

```
    int[] arrayOne = { 13, 3, 1, 192, 4, 6 };
    int[] arrayTwo = new int[6];
```

what does this look like in memory?

how much space does this take up?

how much time does it take to access?

```
    arrayOne[2], arrayOne[5], etc.
```

what happens when you do this?

```
    arrayOne[30] = 56;     // ArrayIndexOutOfBoundsException
    arrayOne.length = 50;  // compiler: array.length cannot be assigned
```

recall:

given a file of fractions (`fractions.txt`), begins with # fractions, each line has numerator then denominator:

```
5
3 4
5 8
-13 16
10 19
5 3
```

write a program that gets the sum of all of the fractions

```java
public static Fraction[] readFractions(Scanner input) {
    int fractionCount = input.nextInt();
    Fraction[] fractions = new Fraction[fractionCount];
    for (int i = 0; i < fractions.length; i++) {
        int num = input.nextInt();
        int den = input.nextInt();
        fractions[i] = new Fraction(num, den);
    }
    return fractions;
}
```

modified problem:

given a file of fractions (`fractions.txt`), each line has numerator then denominator (no # fractions at top):

```
3 4
5 8
-13 16
10 19
5 3
```

write a program that gets the sum of all of the fractions

```java
public static Fraction[] readFractions(Scanner input) {
    int fractionCount = input.nextInt();
    Fraction[] fractions = new Fraction[fractionCount];
    for (int i = 0; i < fractions.length; i++) {
        int num = input.nextInt();
        int den = input.nextInt();
        fractions[i] = new Fraction(num, den);
    }
    return fractions;
}
```

```java
public static Fraction[] readFractions(Scanner input) {
    // int fractionCount = input.nextInt();
    Fraction[] fractions = new Fraction[      ???      ];
    for (int i = 0; i <       ???       ; i++) {
        int num = input.nextInt();
        int den = input.nextInt();
        fractions[i] = new Fraction(num, den);
    }
    return fractions;
}
```

```
public static Fraction[] readFractions(Scanner input) {
    Fraction[] fractions = new Fraction[1000]; // naïve
    int i = 0;                                 // solution
    while (input.hasNext()) {
        int num = input.nextInt();
        int den = input.nextInt();
        fractions[i++] = new Fraction(num, den);
    }
    return fractions;
}
```

problem: don’t know how many fractions the file will have

- hard to create an array of the appropriate size
  (what if >1000 fractions?)
- later parts of the problem are more difficult to solve
  (how many fractions were actually stored in the array?)

luckily, there are other ways to store data besides in an array!

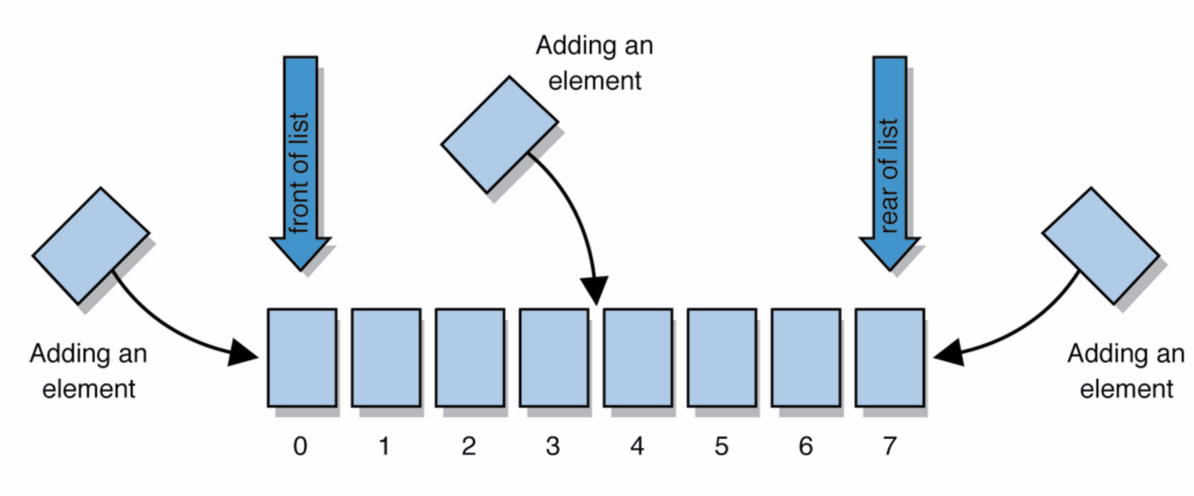
list: a collection storing an ordered sequence of elements

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 list

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 (by default, add to end)

```
[yo, hello, hooray]
```

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
- internally, the list is implemented using an array and a size field

ArrayList

a data structure with:

- contiguous storage of single data type
- compact space requirements
- random access in “constant time”
- variable length

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
clear()	removes all elements of the list
indexOf(**value**)	returns first index where given value is found in list (-1 if not found)
get(**index**)	returns the value at given index
remove(**index**)	removes/returns value at given index, shifting subsequent values to the left
set(**index**, **value**)	replaces value at given index with given value
size()	returns the number of elements in list
toString()	returns a string representation of the list such as "[3, 42, -7, 15]"

ArrayList methods

addAll(**list**)
addAll(**index**, **list**)
	adds all elements from the given list to this list
	(at the end of the list, or inserts them at the given index)

contains(**value**)	returns true if given value is found somewhere in this list
containsAll(**list**)	returns true if this list contains every element from given list
equals(**list**)	returns true if given other list contains the same elements

iterator()
listIterator()
	returns an object used to examine the contents of the list
	(seen later)

lastIndexOf(**value**)	returns last index value is found in list (-1 if not found)
remove(**value**)	finds and removes the given value from this list
removeAll(**list**)	removes any elements found in the given list from this list
retainAll(**list**)	removes any elements *not* found in given list from this list
subList(**from**, **to**)	returns the sub-portion of the list between indexes **from** (inclusive) and **to** (exclusive)
toArray()	returns the elements in this list as an array

type parameters (generics)

syntax:

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

when constructing an `ArrayList`, you must specify the type of elements it will contain between < and >

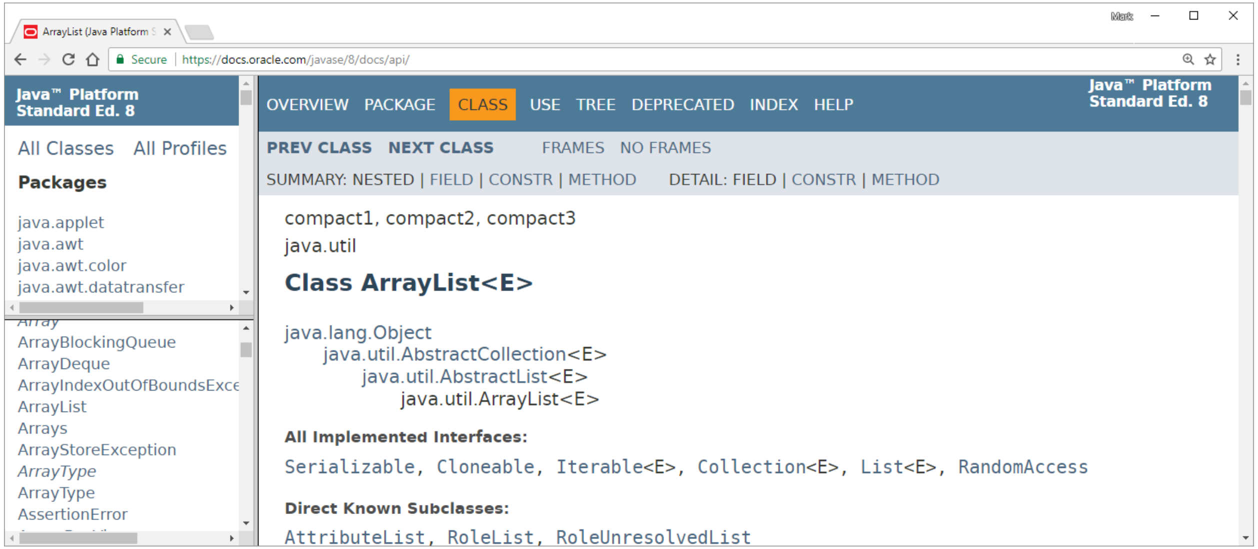
- this is called a type parameter or a generic class
- allows the same `ArrayList` class to store lists of different types

example:

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

learning about classes

the Java API Specification is a huge web page containing documentation about every Java class and its methods



ArrayList (Java Platform S
Mark
Secure | https://docs.oracle.com/javase/8/docs/api/

Java™ Platform
Standard Ed. 8

All Classes All Profiles

Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer

Array
ArrayBlockingQueue
ArrayDeque
ArrayIndexOutOfBoundsExce
ArrayList
Arrays
ArrayStoreException
ArrayType
ArrayType
AssertionError

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

Java™ Platform
Standard Ed. 8

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3
java.util

Class ArrayList<E>

java.lang.Object
    java.util.AbstractCollection<E>
        java.util.AbstractList<E>
            java.util.ArrayList<E>

All Implemented Interfaces:

`Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess`

Direct Known Subclasses:

`AttributeList, RoleList, RoleUnresolvedList`

ArrayList vs. array

construction

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

storing a value

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

retrieving a value

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

`ArrayList` vs. array

doing something to each value that starts with “B”

```
// array
for (int i = 0; i < names.length; i++) {
    if (names[i].startsWith("B")) {
        // ...
    }
}

// ArrayList
for (int i = 0; i < list.size(); i++) {
    if (list.get(i).startsWith("B")) {
        // ...
    }
}
```

ArrayList vs. array

seeing whether the value “Benson” is found

```
// array
for (int i = 0; i < names.length; i++) {
    if (names[i].equals("Benson")) {
        // ...
    }
}

// ArrayList
if (list.contains("Benson")) {
    // ...
}
```

ArrayList as a parameter (syntax):

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

example:

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

you can also return a list (syntax):

```
public static ArrayList<Type> methodName(params) {
```

problem revisited:

given a file of fractions (`fractions.txt`), each line has numerator then denominator (no # fractions at top):

```
3 4
5 8
-13 16
10 19
5 3
```

write a program that gets the sum of all of the fractions

```
public static Fraction[] readFractions(Scanner input) {
    Fraction[] fractions = new Fraction[1000]; // naïve
    int i = 0;                                 // solution
    while (input.hasNext()) {
        int num = input.nextInt();
        int den = input.nextInt();
        fractions[i++] = new Fraction(num, den);
    }
    return fractions;
}
```

```java
public static ArrayList<Fraction> readFractions(Scanner input) {
    ArrayList<Fraction> fractions = new ArrayList<>();

    while (input.hasNext()) {
        int num = input.nextInt();
        int den = input.nextInt();
        fractions.add(new Fraction(num, den));
    }
    return fractions;
}
```

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 special classes called wrapper classes in their place

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

a wrapper is an object whose sole purpose is to hold a primitive value

Primitive Type	Wrapper Type
int	`Integer`
double	`Double`
char	`Character`
boolean	`Boolean`

author: The Come Up Show
https://www.flickr.com/photos/thecomeupshow/28082662994/in/album-72157671052327671/

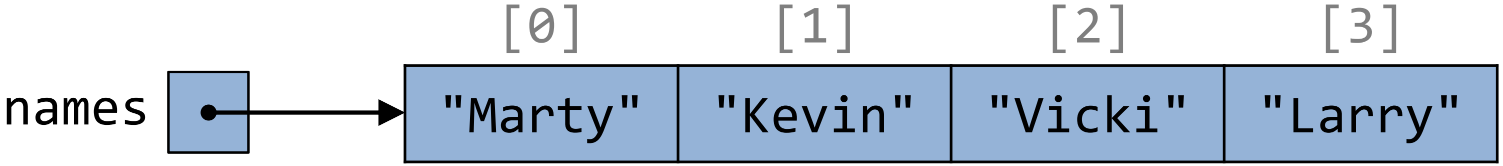
once you construct the list, use it with primitives as normal:

```
ArrayList<Double> grades = new ArrayList<Double>();
grades.add(3.2);
grades.add(2.7);
//...
double myGrade = grades.get(0);
```

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");
System.out.println(names.get(0));       // okay
System.out.println(names.get(3));       // okay
System.out.println(names.get(-1));      // exception
names.add(9, "Aimee");                  // exception
```



ArrayList "mystery"

```
ArrayList<Integer> list = new ArrayList<>();
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);
}
System.out.println(list);
```

answer:

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

ArrayList "mystery" 2

```
ArrayList<Integer> list = new ArrayList<>();
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();
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]
```

an object can have an array, list, or other collection as a field

```
public class Course {
    private double[] grades;
    private ArrayList<String> studentNames;

    public Course() {
        grades = new double[4];
        studentNames = new ArrayList<>();
    }
    // ...
}
```

now each object stores a collection of data inside it

clicker questions

which of the following is the correct syntax to construct an `ArrayList` to store integers?

```
A. ArrayList list = new ArrayList();
B. ArrayList[int] list = new ArrayList[int]();
C. ArrayList list<integer> = new ArrayList<>();
D. ArrayList<Integer> list = new ArrayList<>();
E. ArrayList<Integer> list = new ArrayList();
```

output?

```
int[] x = new int[5];
for (int i = 1; i < x.length; i++) {
    x[i] = x[i - 1] + 1;
}
System.out.printf("%d,%d", x[2], x[4]);
```

A. 3, 5

B. 1, 3

C. 2, 4

D. Error, goes outside of bounds and an exception is thrown.

E. Error, will not compile since array is not initialized.

output?

```
ArrayList<Integer> x
       = new ArrayList<>();
for (int i = 1; i < x.size(); i++) {
    x.set(i, x.get(i-1) + 1);
}
System.out.printf("%d,%d",
        x.get(2), x.get(4));
```

A. 3, 5

B. 1, 3

C. 2, 4

D. Error, goes outside of bounds and an exception is thrown.

E. Error, will not compile since array is not initialized.

next class:

stacks, queues, linked lists