CS1083 Week 2: Arrays, ArrayList

mostly review

David Bremner

2018-01-08

Arrays (1D)

Declaring and using 2D Arrays

2D Array Example

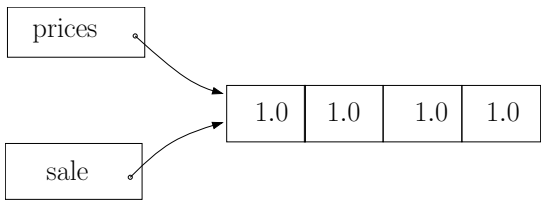
ArrayList and Generics

Multiple references to an array

```
double prices[]={1.0, 1.0, 1.0, 1.0};
double sale[]=prices;

for (int i=0; i< sale.length; i++){
  sale[i] = sale[i]/2;
}

System.out.println(prices[2]);
```



Sale1

Sale2

- Called *aliasing*.
- What is the right way to impliment this?

Bounds Checking

In Java (unlike C or C++), `price[i]` is equivalent to

```
1 if (i < 0 || i>= price.length) {
2   throw new ArrayOutOfBoundsException();
3 } else {
4   return element i from array price
5 }
```

- For the moment, line 2 means "Crash"
- What is the result of `prices[prices.length]`

Passing arrays

What does this program display?

ArrayStatic

```
public class ArrayStatic{
    static void s(int t[]) {
        t[0] = 3; t[0] = 6; t[2] = 9;
    }
    public static void main(String[] args){
        int t[]={0, 1, 2};
        t[0] = 1; t[1] = 2; t[2] = 3;
        s(t);
        for (int i = 0; i < 3; i++)
            System.out.print(t[i] + " ");
    }
}
```

partially filled arrays

Goals

- keep track of list of scores
- compute average

partially filled arrays

Goals

- keep track of list of scores
- compute average

Scores1

```
private int count;
private double[] scores;

public Scores1(int max) {
    count = 0;
    scores = new double[max];
}
```

Adding elements

```
public void add(double score) {
    scores[count] = score;
    count++;
}
```

Scores1

encapsulation

- Why a mutator / private instance variables?
- What could go wrong if we made them public?

Computing the average

```
public double getAverage() {
    double sum=0.0;
    for (int i=0; i<count; i++){
        sum += scores[i];
    }
    return sum/count;
}
```

class design

- Is it reasonable to call this an accessor? Why, or why not?
- How could we make this method faster while preserving the same API?

Encapsulation and class design

- Encapsulation is about a well defined API
- The designer of a library controls what operations are permitted
- This is necessary for the library designer to guarantee correctness
- Generally make instance variables private and provide getters and setters.
- Mutators allow sanity checking input.

Arrays (1D)

Declaring and using 2D Arrays

2D Array Example

ArrayList and Generics

2D Arrays

- Recall that you can declare an array of *anything*.
- Even an array of *arrays*

```
int [][] grid;
```

- As usual, the array of arrays must be created

```
grid=new int[10][]
```

- And each element (which is in turn an array) must also be created.

```
for (int i=0; i<10; i++)
   grid[i]=new int[i+1];
```

2D Arrays

- Recall that you can declare an array of *anything*.
- Even an array of *arrays*

```
int [][] grid;
```

- As usual, the array of arrays must be created

```
grid=new int[10][]
```

- And each element (which is in turn an array) must also be created.

```
for (int i=0; i<10; i++)
   grid[i]=new int[i+1];
```

2D Arrays

- Recall that you can declare an array of *anything*.
- Even an array of *arrays*

```
int [][] grid;
```

- As usual, the array of arrays must be created

```
grid=new int[10][]
```

- And each element (which is in turn an array) must also be created.

```
for (int i=0; i<10; i++)
   grid[i]=new int[i+1];
```

2D Arrays

- Recall that you can declare an array of *anything*.
- Even an array of *arrays*

```
int [][] grid;
```

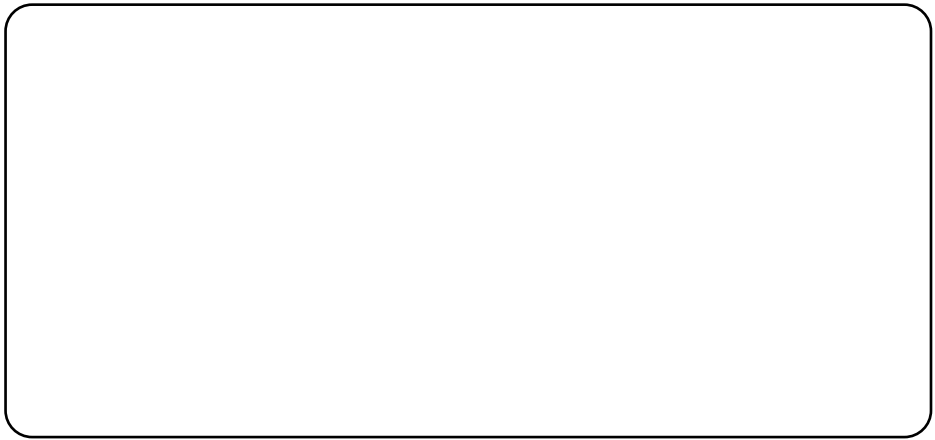
- As usual, the array of arrays must be created

```
grid=new int[10][]
```

- And each element (which is in turn an array) must also be created.

```
for (int i=0; i<10; i++)
   grid[i]=new int[i+1];
```

- And then can be used



- What is the output?

- And then can be used

```
for (int i=0; i<10; i++) {




    System.out.println();
}
```

- What is the output?

- And then can be used

```
for (int i=0; i<10; i++) {
    for (int j=0; j<grid[i].length; j++) {
        if (j>0)

        System.out.print(grid[i][j]);
    }
    System.out.println();
}
```

- What is the output?

- And then can be used

```
for (int i=0; i<10; i++) {
    for (int j=0; j<grid[i].length; j++) {
        if (j>0)
            System.out.print(" ");
        System.out.print(grid[i][j]);
    }
    System.out.println();
}
```

- What is the output?

- And then can be used

```
for (int i=0; i<10; i++) {
    for (int j=0; j<grid[i].length; j++) {
        if (j>0)
            System.out.print(" ");
        System.out.print(grid[i][j]);
    }
    System.out.println();
}
```

- What is the output?

The "normal case"

- In general, every row is the same length, and we can take the shortcut

```
grid=new int[10][10];
```

- And then the same code yields:

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

The "normal case"

- In general, every row is the same length, and we can take the shortcut

```
grid=new int[10][10];
```

- And then the same code yields:

0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

Reading PBM Files

```
P1
3 3
111
101
111
```

- PBM Format
- Using Scanner: nextInt, next
- Read header and initialize
- Read bits

Reading PBM Files

```
P1
3 3
111
101
111
```

- PBM Format
- Using Scanner: nextInt, next
- Read header and initialize
- Read bits

Reading PBM Files

```
P1
3 3
111
101
111
```

- PBM Format
- Using Scanner: nextInt, next
- Read header and initialize
- Read bits

Reading PBM Files

```
P1
3 3
111
101
111
```

- PBM Format
- Using Scanner: nextInt, next
- Read header and initialize
- Read bits

Reading PBM Files

```
P1
3 3
111
101
111
```

- PBM Format
- Using Scanner: nextInt, next
- Read header and initialize
- Read bits

Instance Variables

PBM

```
public class PBM {
    private int[][] bits;
    private int rows,columns;
    ⋮
}
```

Constructor

```
PBM() throws IOException{
    Scanner sc =
        new Scanner(System.in);
```

Constructor

```
PBM() throws IOException{
    Scanner sc =
        new Scanner(System.in);
    if (!sc.next().equals("P1"))
        throw new IOException("Format error");
```

Constructor

```
PBM() throws IOException{
    Scanner sc =
        new Scanner(System.in);
    if (!sc.next().equals("P1"))
        throw new IOException("Format error");
    columns=sc.nextInt();
    rows=sc.nextInt();
```

Constructor

```
PBM() throws IOException{
    Scanner sc =
        new Scanner(System.in);
    if (!sc.next().equals("P1"))
        throw new IOException("Format error");
    columns=sc.nextInt();
    rows=sc.nextInt();
    bits=new int[rows][columns];
```

Constructor

```
PBM() throws IOException{
    Scanner sc =
        new Scanner(System.in);
    if (!sc.next().equals("P1"))
        throw new IOException("Format error");
    columns=sc.nextInt();
    rows=sc.nextInt();
    bits=new int[rows][columns];
    for (int i=0; i< rows; i++){
        String line=sc.next();



    }
```

Constructor

```
PBM() throws IOException{
    Scanner sc =
        new Scanner(System.in);
    if (!sc.next().equals("P1"))
        throw new IOException("Format error");
    columns=sc.nextInt();
    rows=sc.nextInt();
    bits=new int[rows][columns];
    for (int i=0; i< rows; i++){
        String line=sc.next();
        for (int j=0; j<columns; j++)

    }
```

Constructor

```
PBM() throws IOException{
    Scanner sc =
        new Scanner(System.in);
    if (!sc.next().equals("P1"))
        throw new IOException("Format error");
    columns=sc.nextInt();
    rows=sc.nextInt();
    bits=new int[rows][columns];
    for (int i=0; i< rows; i++){
        String line=sc.next();
        for (int j=0; j<columns; j++)
            bits[i][j]=line.charAt(j)-'0';
    }
```

toString

```
public String toString(){
    String result="";









    return result;
```

toString

```
public String toString(){
    String result="";
    for (int i=0; i< rows; i++){









        result +="\n";
    }
    return result;
```

toString

```
public String toString(){
    String result="";
    for (int i=0; i< rows; i++){
        for (int j=0; j<columns; j++){









        }
        result +="\n";
    }
    return result;
```

toString

```
public String toString(){
    String result="";
    for (int i=0; i< rows; i++){
        for (int j=0; j<columns; j++){
            if (bits[i][j]==1)
                result += "*";
            else
                result += " ";
        }
        result +="\n";
    }
    return result;
```

ArrayList class

- In `java.util`
- Stores list of objects
- Allows objects to be added and removed
- Implemented using an array
  - Resized as needed
- Uses generic type

Generics

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

- Type of elements in collections specifed in <>
- Only objects of specifed type can be added to collection
- Objects removed already have correct type (don't need to cast)
- For primitive types use wrapper class (e.g. Integer)

Working with ArrayList

Create list of objects of Passenger class, which stores passenger name, id, weight, and luggage weight.

```
ArrayList<Passenger> list = new ArrayList<Passenger>();
list.add(new Passenger("Pete",50,20));
list.add(new Passenger("Fred",80,30));
```

Iterating Through List

Could do like this:

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

- Better to use for-each loop

For-Each Loop

- Works with objects that implement Iterable interface
  - We'll see interfaces soon
  - ArrayList implements Iterable
  - Also works with regular arrays
- For processing all elements of collecton (list or array)

For-Each Loops

```
for(Passenger p : list){
    System.out.println(p);
}
int[] a = {2, 13, -6, 8};
int sum = 0;
for(int val : a){
    sum += val;
}
```

Variable Length Parameter Lists

If a small number of objects of same type to be passed to method, don't need to use array

DL6.5

```
mean1 = average(42, 69, 37);
mean2 = average(35,43,93,40,21,75);
```

Variable Length Parameter Lists

```
public double average(int ... list){
    double result = 0.0;
    if(list.length != 0){
        int sum = 0;
        for(int num : list)
            sum += num;
        result = (double)sum/list.length;
        return result;
    }
}
```

Parameters automatically put into array