

Arrays; Object References

Atul Prakash

Grid in Assignment 2

- How do you represent the state of the game so that you can print it out after each step?
- Need a way to represent 1-dimensional or 2-dimensional grids
- Solution: arrays

Arrays

- Arrays are simply a fixed-length collection of objects, indexed by a number.
- To create an array of 10 integers, one can do:
 - `int[] x; // declares x to be an array type.`
 - `x = new int[10]; // creates an int array of size 10.`
 - `x[0], x[1], ..., x[8], x[9]`: the ten elements

Array usage

```
int[] nums;
```

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

```
nums[0] = 6;
```

```
nums[1] = 10;
```

```
nums[2] = nums[0] + nums[1];
```

```
nums[3] = nums[1];
```

nums



0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

6	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

6	10	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

6	10	16	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

6	10	16	10	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

Example

```
int nums[];  
nums = new int[10];  
  
for (int i = 0; i < 10; i++) { // Initialize the array with squares  
    nums[i] = i*i;  
}  
  
for (int i = 0; i < 10; i++) { // print out the array  
    System.out.printf("index = %d, array content = %d\n", i, nums[i]);  
}
```

0	1	4	9	16	25	36	49	64	81
0	1	2	3	4	5	6	7	8	9

```
index = 0, cell content = 0  
index = 1, cell content = 1  
index = 2, cell content = 4  
index = 3, cell content = 9  
index = 4, cell content = 16  
index = 5, cell content = 25  
index = 6, cell content = 36  
index = 7, cell content = 49  
index = 8, cell content = 64  
index = 9, cell content = 81
```

printf: Formatted output

%d: substituted by the corresponding integer argument

Two-dimensional arrays

- We want to create a 10x20 array of 1's.
- Java (like C) only can create one-dimensional arrays.
- Solution: Create an array of arrays.

```
// create an array or arrays
int[][] nums;

// create 10 arrays. nums[0]...nums[9] will
// be of type int[].
nums = new int[10][];

// make each element a 20-element array
for (int i = 0; i < 10; i++) {
    nums[i] = new int[20];
}

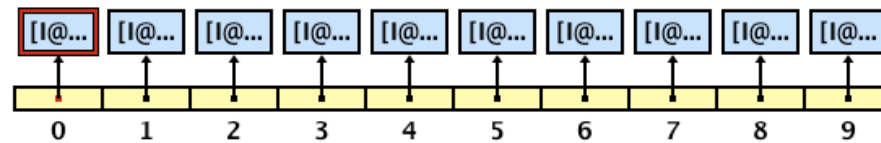
// fill the array with ones.
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 20; j++) {
        nums[i][j] = 1;
    }
}
```

Visualization in jGrasp

nums

Type `int[][]` [Current] View Presentation

Width 4.0 Scale 1.0



nums [0]

Type `int[]` [Current] View Presentation

Width 4.0 Scale 1.0

1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

To see Visualizations in jGrasp -- see Tutorial



The screenshot shows the jGRASP IDE interface. The main window displays the source code for `Arraydemo.java`. The code is as follows:

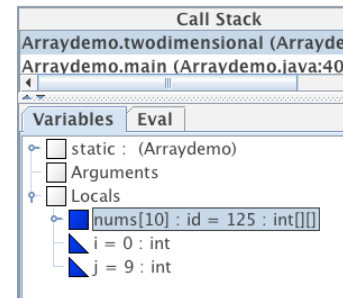
```
19     int[][] nums;  
20     // create 10 arrays  
21     nums = new int[10][];  
22     // make each element a 20-element array  
23     for (int i = 0; i < 10; i++) {  
24         nums[i] = new int[20];  
25     }  
26     // fill the array with ones.  
27     for (int i = 0; i < 10; i++) {  
28         for (int j = 0; j < 20; j++) {  
29             nums[i][j] = 1;  
30         }  
31     }  
32 }  
33  
34 /**  
35  * @param args  
36  */  
37 public static void main(String[] args) {  
38     // TODO Auto-generated method stub
```

The debugger is currently paused at line 29. The left sidebar shows the following information:

- Threads:** No threads are currently running.
- Call Stack:** Shows the current method call stack:
 - `Arraydemo.twodimensional (Arraydemo.java)`
 - `Arraydemo.main (Arraydemo.java:40) pc =`
- Variables:** Shows the current state of variables:
 - `static : (Arraydemo)`
 - `Arguments`
 - `Locals`
 - `nums[10] : id = 125 : int[][]`
 - `i = 0 : int`
 - `j = 9 : int`

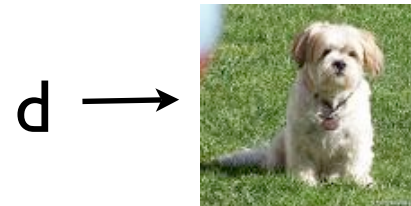
jGrasp: Visualization steps

- Set the debug mode under "Build"
- Do Build->Compile (or hit the )
- Set a breakpoint at the line where you want to start viewing the data structure
- Build->Debug (or the )
- The program will pause at the breakpoint
- In the Variables window on the left, click on a variable and drag it anywhere. The visualization will open. Step through the code and see the value change.



Object References

- Dog d;
 - d is a **reference** to a dog. It is not the actual dog.
- d = **null**;
 - **null** is a special object to help initialize references. Like 0.
- d = new Dog("Fido", "woof");
 - d now refers to a dog object



Assigning Object References

- Dog d1, d2;
- d2 = d1
- Creates two object references
- Only copies the reference, not the object
- `d1 = new Dog("Fido", "woof");`

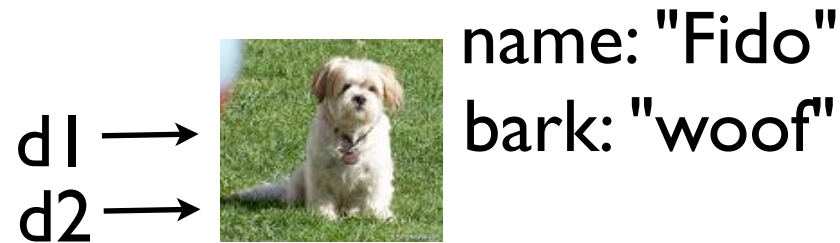
d1 →



d1 →
d2 →



Aliasing of references



- `d1.setBark("huff");`



- What are `d1.getBark()` and `d2.getBark()`?

In Java...

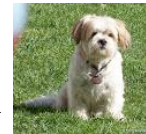
- Most of the variables are references to objects.
- Array variables are always references to array objects.
- Exceptions: primitive types, such as
 - int, boolean, byte, short, long, float, double

Example: Primitive Types vs. object types

```
public static void main(String[] args) {  
    int i, j; // Not references. Basic types. i = 0, j = 0  
    i = 2;    // i = 2, j = 0  
    j = i;    // i = 2, j = 2.  
    i = 3;    // i = 3, j = 2  
    System.out.println(j); // will print 2, not 3.
```

```
    Dog d1, d2; // References.  
    d1 = new Dog("Fido", "woof");  
    d2 = d1;  
    d1.setBark("huff");  
    d2.bark(); // will print "huff", not "woof"
```

d1 →
d2 →



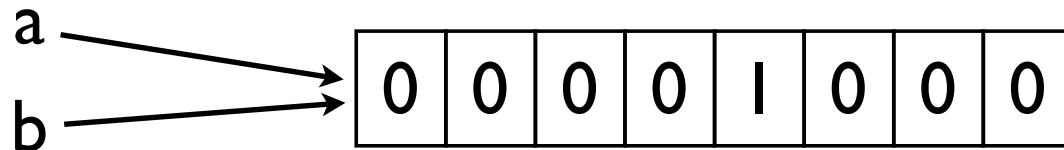
name: "Fido"
bark: "huff"

```
}
```

*Java convention: Types starting with **small cap** (e.g., `int`) are **primitive**. Others should start with a capital letter (e.g., `String`, `Dog`) and are object types.*

Assigning Arrays

```
// Assigning array references.  
int[] a;  
int [] b;  
a = new int[8];  
b = a; // reference copy. a and b refer to the same array  
a[4] = 3;  
System.out.println(b[4]); // Will print 3.
```



Arrays of Objects

```
// Arrays of objects
Dog[] dogarray; // Create a reference to an array
dogarray = new Dog[3]; // Create 3 references to dogs

// Create the dogs
dogarray[0] = new Dog("Fido", "woof");
dogarray[1] = new Dog("Daisy", "huff");
dogarray[2] = new Dog("Ginger", "woof");
```

```
ObjectReferences.java /Users/aprakash/Docu

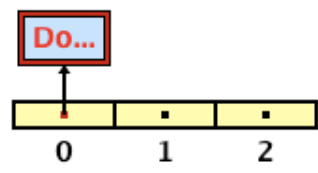
Dog d1, d2;
d1 = new Dog("Fido", "woof");
d2 = d1;
d1.setBark("huff");
d2.bark(); // will print "huff", not "woof"

// Assigning array references.
int[] a;
int [] b;
a = new int[10];
b = a; // reference copy. a and b refer to the same array.
a[4] = 3;
System.out.println(b[4]); // Will print 3.

// Arrays of objects
Dog[] dogarray;
dogarray = new Dog[3];
dogarray[0] = new Dog("Fido", "woof");
dogarray[1] = new Dog("Daisy", "huff");
dogarray[2] = new Dog("Ginger", "woof");
```

Ref...

er=NONE -Xdebug -Xrunjdw:transport=dt_socket,suspend=y,s



dogarray [0]

Type Dog [Current] View Basic

Accessibility Context ObjectReferences [Current]

Show Inaccessible Fields Sort By Natural Order

- id = 125 : Dog
 - name = "Fido" : id = 378 : java.lang.String : Dog.name
 - bark = "woof" : id = 379 : java.lang.String : Dog.bark