Arrays

Atul Prakash

Readings: Chapter 10, Downey

Sun's Java tutorial on Arrays:

http://java.sun.com/docs/books/tutorial/java/nutsandbolts/arrays.html

Grid in Assignment 2

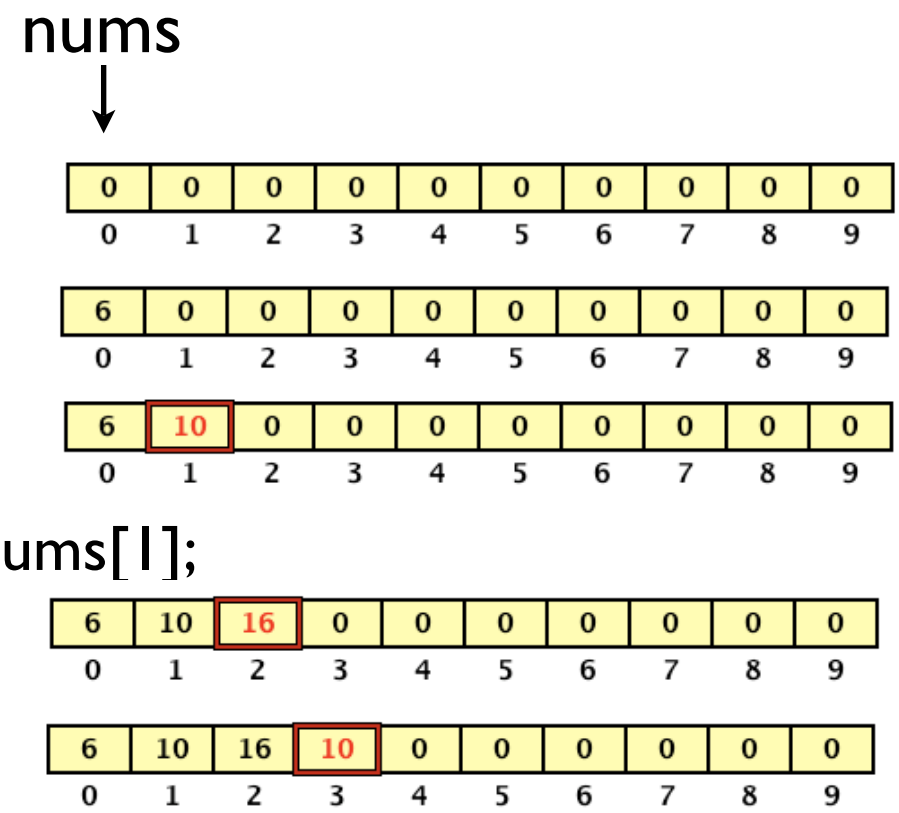
- How do you represent the state of the game so that you can print it out after each step?
- Need a way to reprsent 1-dimensional or 2-dimensional grids
- Solution: arrays

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

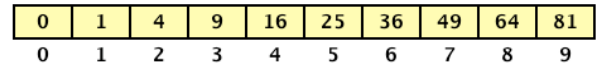
Array usage

```
int[] nums;

nums = new int[10];

nums[0] = 6;

nums[1] = 10;

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

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



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]);
}
```




```
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

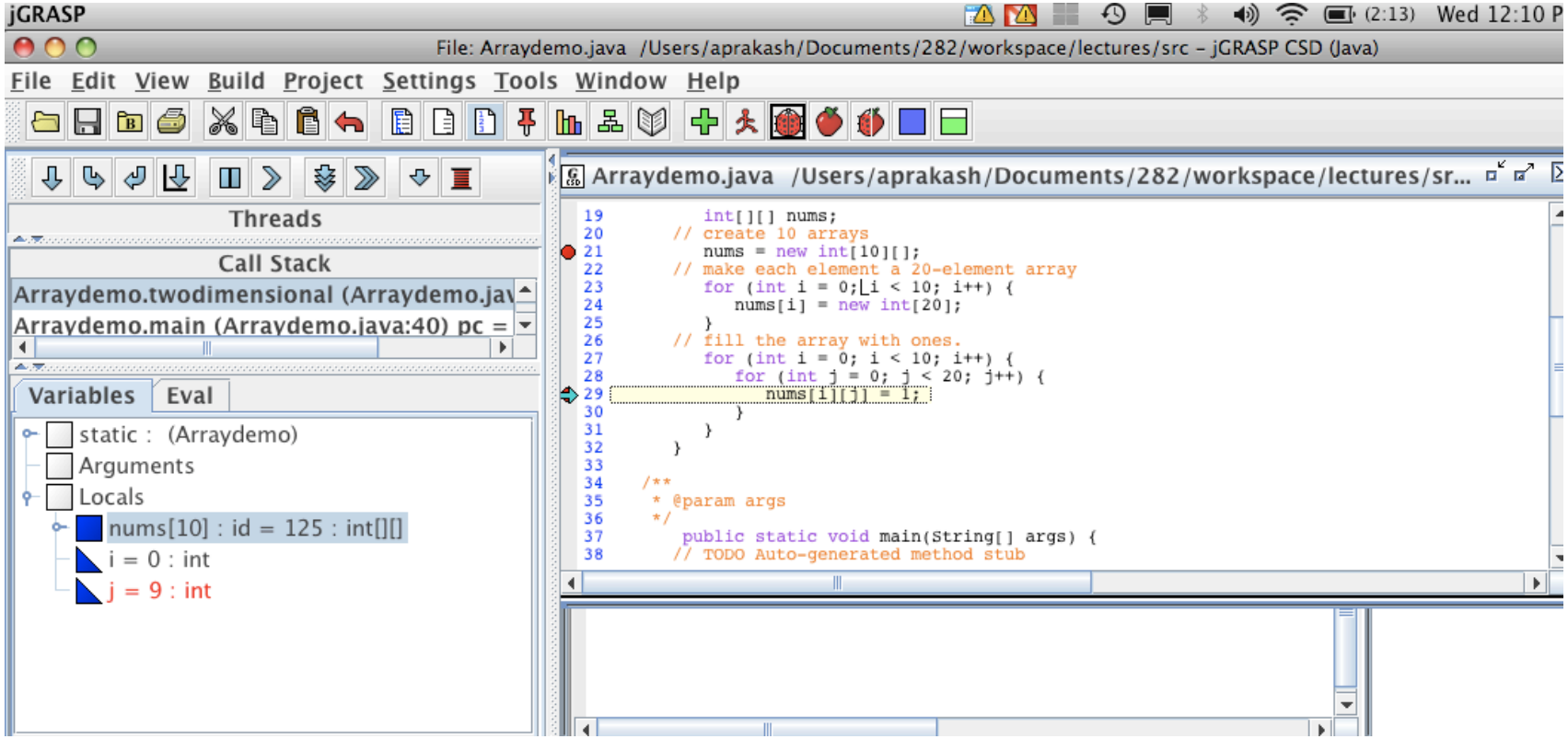
%d: substituted by the corresponding integer argument

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.

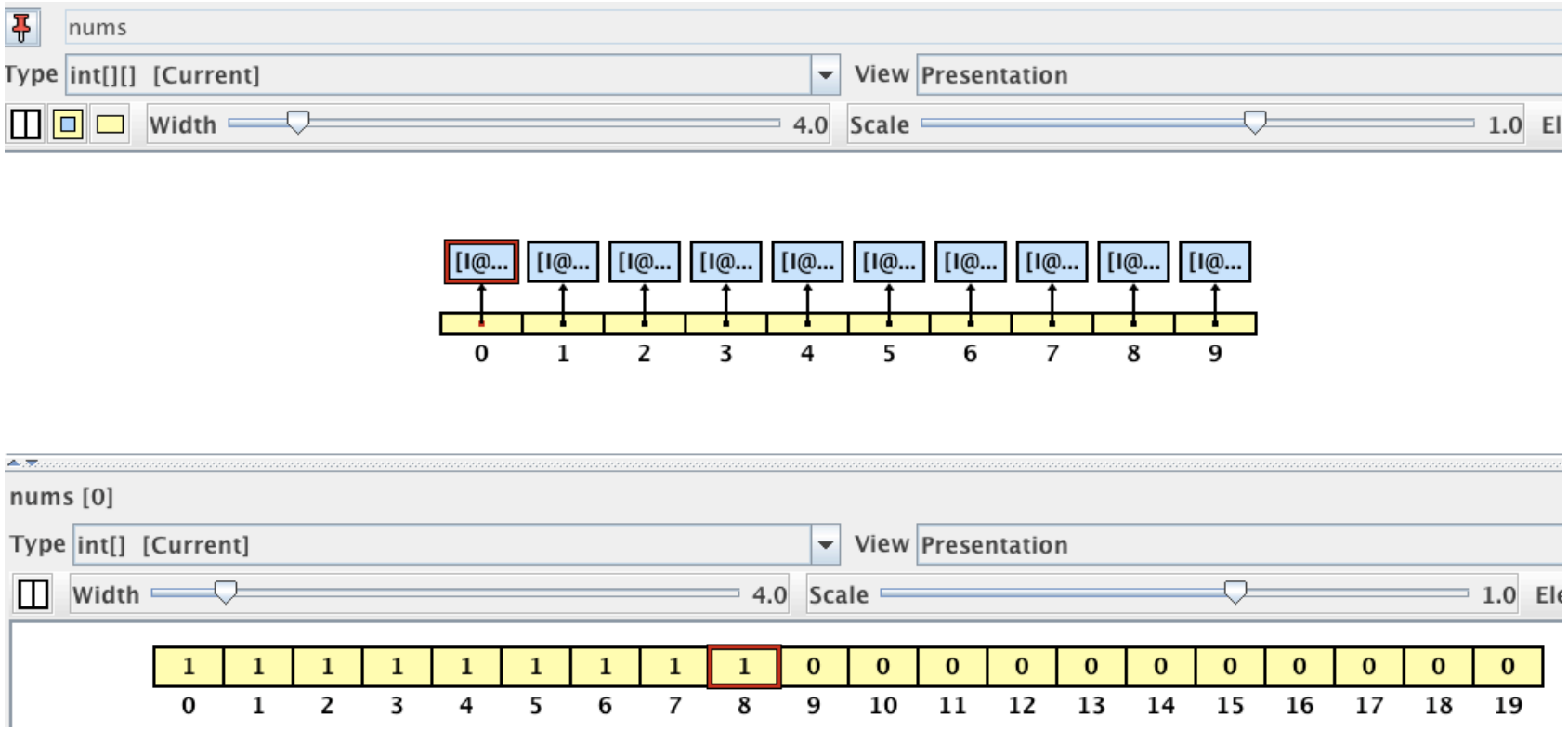
```java
// create an array or arrays
int[][] nums;

// 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 of Arrays



Courtesy: jGrasp Editor

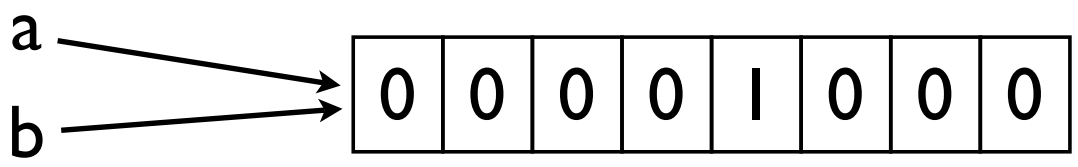
Visualization



Courtesy: jGrasp Editor

Assigning Arrays - Aliasing occurs

```
// 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.
```



Bound Checking

- Java does safety checks on arrays bounds.
  - Exception occurs if array index is negative or >= size of the array
- Different from C++, where array bounds are not checked one can write past an array, trashing a program
- Different from Python lists: negative indices not allowed

Arrays versus Lists

- Arrays look like lists of values and can sometimes be used interchangeably, but some differences

Arrays (as in Java)	Lists (e.g., Python lists)
fixed-length	variable-length
Fast for random access. Any element can be accessed fast	Fast for sequential access. Random access could be slow
Not designed for fast insert/ deletes in the middle of the array. Would require shifting elements to permit indexing	Designed for fast inserts/deletes, typically

- Java has lists as well: ArrayList , LinkedList, Vector. More on that later.

Stop here

- The following slides will make more sense after we discuss objects

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

d → 

Assigning Object References

- Dog d1, d2;
  - Creates two object references
- d1 = new Dog("Fido", "woof");

d1 ⟶ 

- d2 = d1
  - Only copies the reference, not the object

d1 ⟶
d2 ⟶ 

Aliasing of references

d1 ⟶
d2 ⟶

name: "Fido"
bark: "woof"

- d1.setBark("huff");

d1 ⟶
d2 ⟶

name: "Fido"
bark: "huff"

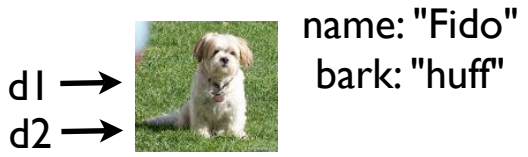
- What are d1.getBark() and d2.getBark()?

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

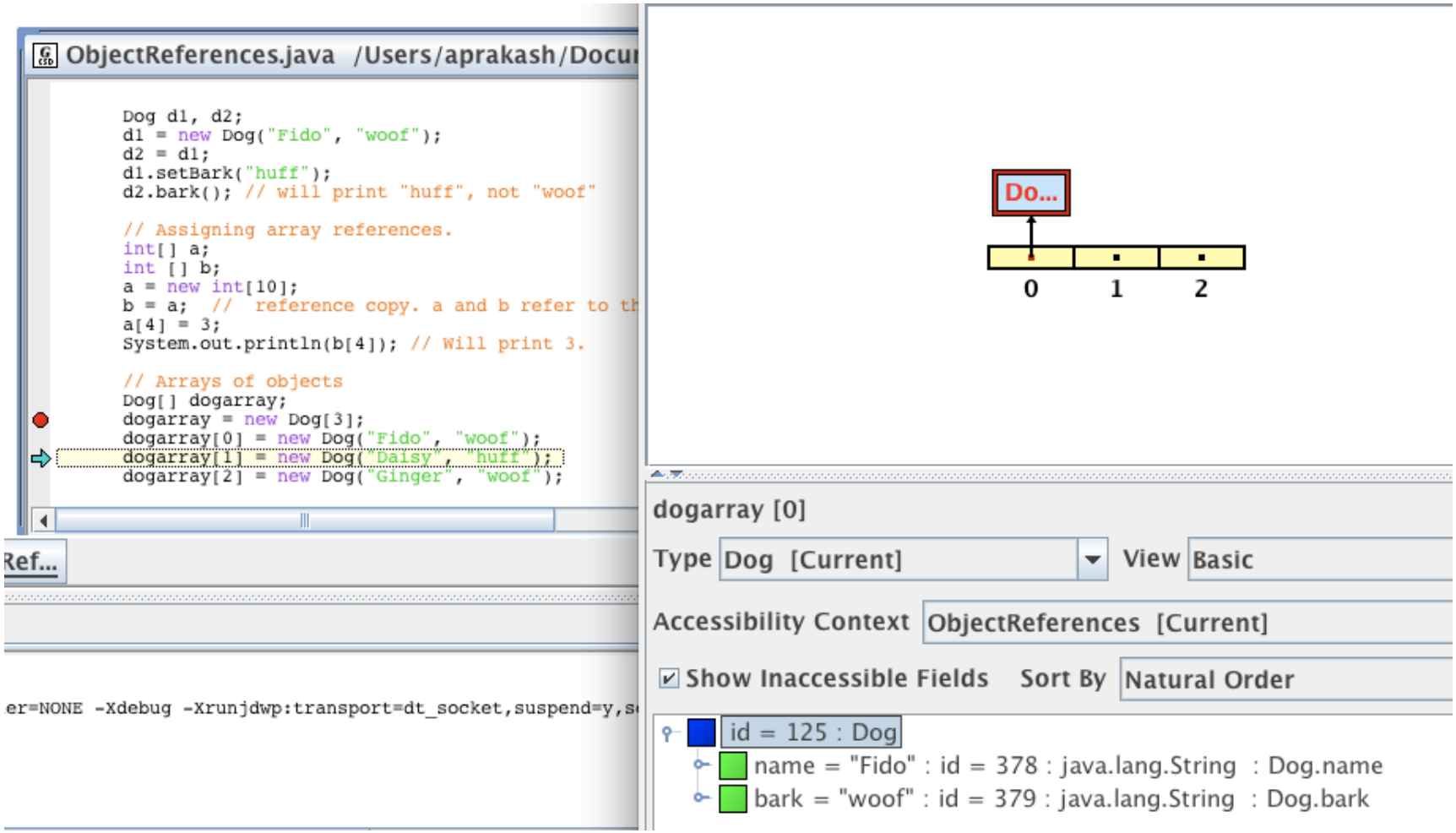
```
public static void main(String[] args) {
    int i, j; // Not references. Basic types. i = 0, j = 0
    i = 2;    // i = 2, j = 0
    j = i;    // i = 2, j = 2.
    i = 3;    // i = 3, j = 2
    System.out.println(j); // will print 2, not 3.

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


}
```



*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.*

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 th
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 -Xrunjdwp:transport=dt_socket,suspend=y,s
Do...
0
1
2
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