Exam 2 Review
Objects, Arrays, Strings

Objects

- Defined by template given in as class statement.
- An object is created by invoking the class's constructor using the new keyword.
- An objects is stored in a variable declared with class as type
- Values passed to a constructor must be copied to object fields to "stick" … why?

```
Tree myMaple;  // Variable defined as type Tree

void setup() {
  myMaple = new Tree("maple", 30.3);   // Create
}

class Tree {
  String name;
  float height;

  Tree( String tname, float theight) {
    name = tname;
    height = theight;
  }

  void draw() {
    fill( 0, 255, 0 );
    ellipse(random(width),random(height),50,50);
  }
}
```

fields

constructor

method

Creating Objects

1. Declare a variable with the class as type
2. Invoke the constructor using the new keyword and assign to variable

```
Tree myMaple;              // Variable defined as type Tree

myMaple = new Tree("maple", 30.3);   // Create and assign


// -----

// Two steps combined in one
Tree myMaple = new Tree("maple", 30.3);
```

Creating Objects

- What is wrong with this?

```
Tree myMaple;              // Variable defined as type Tree

void setup() {
  Tree myMaple = new Tree("maple", 30.3);   // Combined
}
```

Using Objects

- variable :: fields (variable inside an object)
- function :: method (function inside an object)

- An variable that stores an object is used to scope access to the fields and methods of that particular object

Using Objects

```
Tree myMaple;

void setup() {
  myMaple = new Tree("maple", 30.3);
}

void draw() {
  myMaple.draw();
}

class Tree {
  String name;
  float height;

  Tree( String tname, float theight) {
    name = tname;
    height = theight;
  }

  void draw() {
    fill( 0, 255, 0 );
    rect( 10, 10, 50, 300 );
  }

}
```

Using Objects

What is wrong with this?

```
Tree myMaple;

void setup() {
  myMaple = new Tree("maple", 30.3);
}

void draw() {
  Tree.draw();
}

class Tree {
  String name;
  float height;

  Tree( String tname, float theight) {
    name = tname;
    height = theight;
  }

  void draw() {
    fill( 0, 255, 0 );
    rect( 10, 10, 50, 300 );
  }

}
```

Arrays - Creating

- A structure that can hold multiple items of a common data type
- Arrays can hold any data type, including objects
- The data type to be held by an array must be declared as part of the array declaration
- Arrays are themselves a kind of type, which is made by adding brackets to the type that the array can hold

Arrays – Creating and Init'ng (3 Steps)

1. Declare an array variable
   - The variable is NOT an array
2. Create an array and assign it to the variable
   - Use the new keyword and size
   - The array is filled with default values
     - int <- 0
     - float <- 0.0
     - boolean <- false;
     - any object including String <- null
3. Fill the array with items of appropriate type

```
Tree[] trees;
```

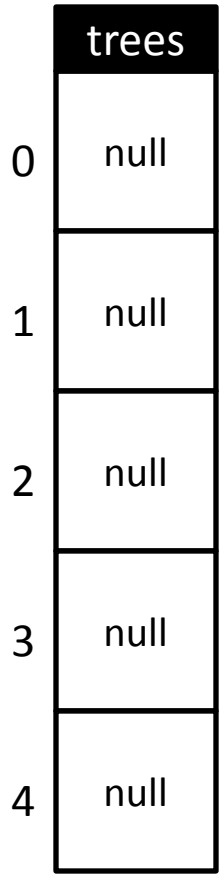
Step 1

trees

← No array. Only a variable that can hold an array.

```
Tree[] trees;
trees = new Tree[5];
```

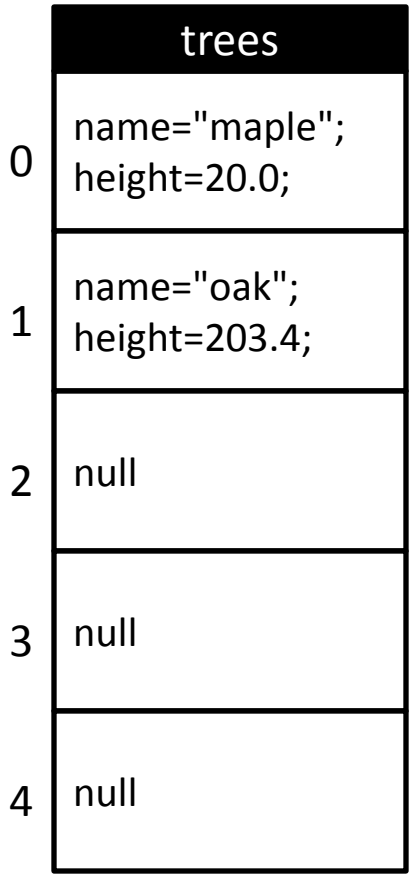
Step 2



← An empty array. null Tree objects.

```
Tree[] trees;
trees = new Tree[5];
trees[0] = new Tree("maple", 20.0);
trees[1] = new Tree("oak", 203.4);
```

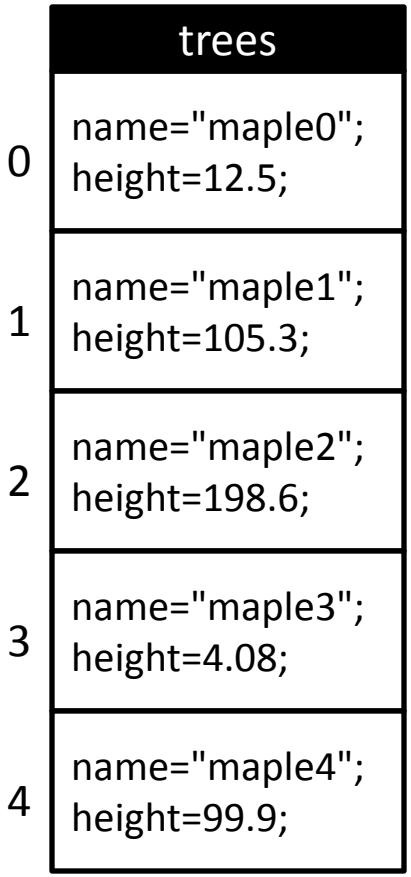
Step 3



← An array with two Tree objects.

```
Tree[] trees;
trees = new Tree[5];
for (int i=0; i<5; i++) {
  trees[i] = new Tree( "maple"+i, random(200.0) );
}
```

Step 3



← An array with five Tree objects.

```
int[] ages;
```

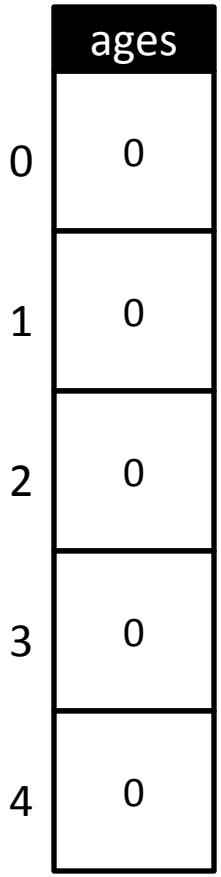
Step 1

ages

← No array. Only a variable that can hold an array.

```
int[] ages;
ages = new int[5];
```

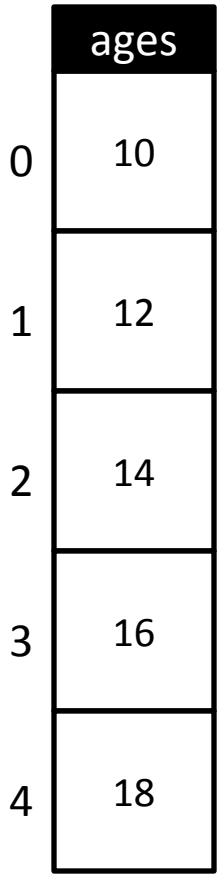
Step 2



← An empty array. Default ints (0).

```
int[] ages;
ages = new int[5];
for (int i=0; i<5; i++) {
  ages[i] = 10 + 2*i;
}
```

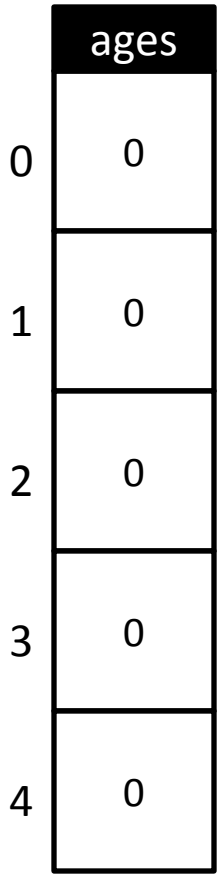
Step 3



← An array with five integers.

```
int[] ages = new int[5];

// Same as
// int[] ages;
// ages = new int[5];
```

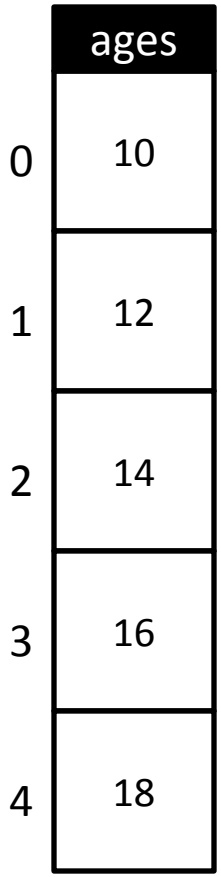
Step 1+2



← An empty array. Default ints (0).

```
int[] ages = new int[] {10, 12, 14, 16, 18};

// Same as
// int[] ages = new int[5];
// for (int i=0; i<5; i++) { ages[i] = 10 + 2*i; }
```

Step 1+2+3



← An array with five integers.

Arrays – Using

- An item in an array is accessed by following an array variable with square brackets containing the item number (index)
- The result of the array accessor expression is the item in the array at the index
- Array indexes start with 0
- Once accessed with brackets, the result can be used as if it was the item at the location in the array

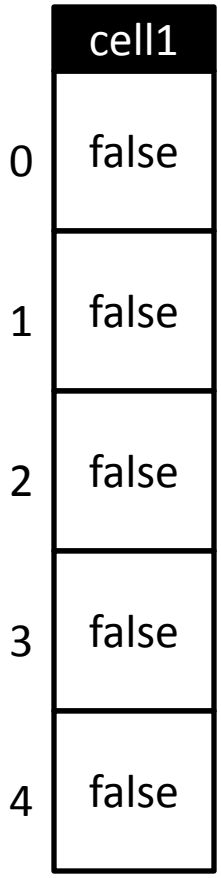
```
Tree[] trees;

void setup() {
  trees = new Tree[3];
  trees[0] = new Tree("maple", 30.3);
  trees[1] = new Tree("oak", 130.3);
  trees[2] = new Tree("spruce", 230.3);
}

void draw() {
  for (int i=0; i<trees.length; i++ ) {
    trees[i].draw();
  }
}

class Tree {
  String name;
  float height;

  Tree( String tname, float theight) {
    name = tname;
    height = theight;
  }

  void draw() {
    fill( 0, 255, 0 );
    ellipse( random(width), random(height), 50, 50 );
  }
}
```

Arrays of arrays (2D Arrays)

- If an array can be made of any type by adding brackets, and …
- an array is a kind of type, then …
- an array of arrays should be possible by adding a second set of brackets

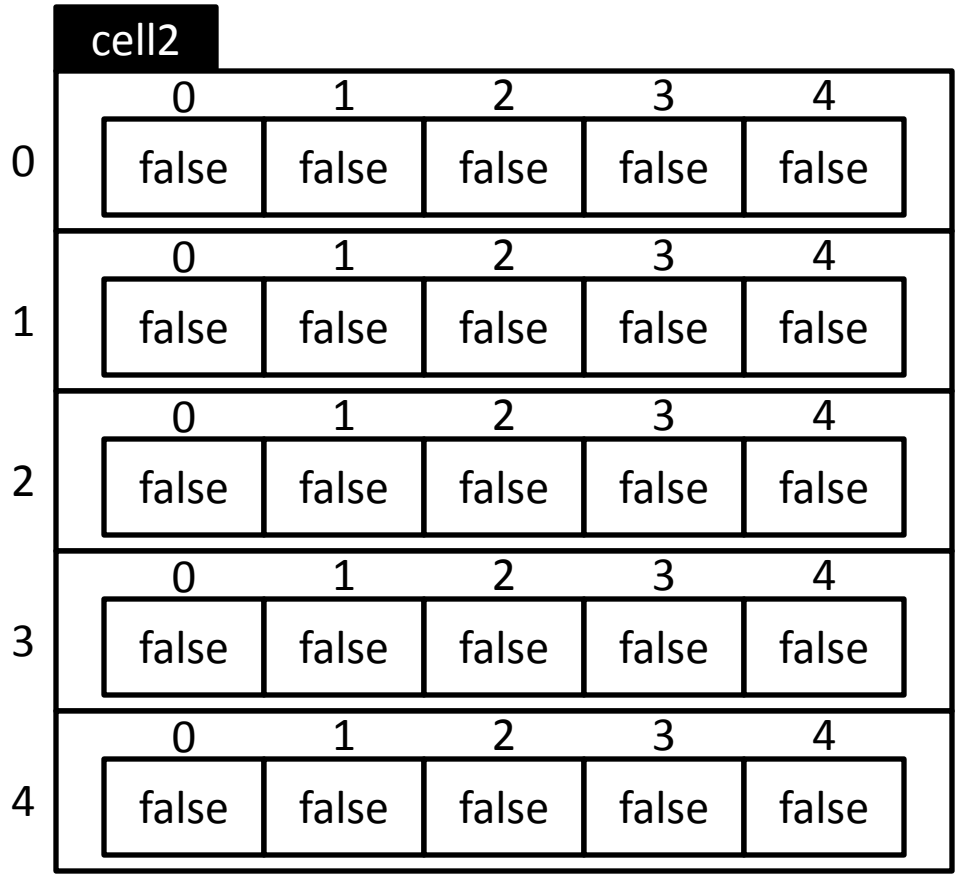
```
boolean[] cell1;     // A variable that holds an array of booleans

boolean[][] cell2;   // A variable that holds an array of
                     // boolean arrays
```

```
boolean[] cell1;
cell1 = new boolean[5];
```



← One-dimensional array

```
boolean[][] cell2;
cell2 = new boolean[5][5];
```



← Two-dimensional array

… an array of arrays

```
boolean[][] cell2;
cell2 = new boolean[5][5];

cell2[1][2] = true;
```

cell2	0	1	2	3	4
0	false	false	false	false	false
1	false	false	**true**	false	false
2	false	false	false	false	false
3	false	false	false	false	false
4	false	false	false	false	false

Proving a 2D array is an array of arrays

- Access fields and methods of top-level array

```
void setup() {

  boolean[][] cell2;
  cell2 = new boolean[5][5];   // Create array of arrays

  println( cell2[0].length );  // Access array

  cell2[1][2] = true;          // Access array in array
  println( cell2[1] );         // Access array
}
```

```
5
[0] false
[1] false
[2] true
[3] false
[4] false
```

Proving a 2D array is an array of arrays

- Build a "ragged array"

```
void setup() {

  boolean[][] cell2;
  cell2 = new boolean[5][];

  cell2[0] = new boolean[2];
  cell2[1] = new boolean[4];
  cell2[2] = new boolean[1];

  println("---");
  println(cell2[0]);
  println("---");
  println(cell2[1]);
  println("---");
  println(cell2[2]);
  println("---");
  println(cell2[3]);
  println("---");
  println(cell2[4]);
}
```

```
---
[0] false
[1] false
---
[0] false
[1] false
[2] false
[3] false
---
[0] false
---
null
---
null
```

Making Strings

- Declaring String objects with no chars

```
String myName;
String myName = new String();
```

- Declaring String objects init'd w/ char array

```
String myName = "Fred";
String myName = new String("Fred");
```

String class methods

- `charAt(`*`index`*`)`
  - Returns the character at the specified index
- `equals(`*`anotherString`*`)`
  - Compares a string to a specified object
- `equalsIgnoreCase(`*`anotherString`*`)`
  - S/A ignoring case (i.e. 'A' == 'a')
- `indexOf(`*`char`*`)`
  - Returns the index value of the first occurrence of a character within the input string
- `length()`
  - Returns the number of characters in the input string
- `substring(`*`startIndex, endIndex`*`)`
  - Returns a new string that is part of the input string
- `toLowerCase()`
  - Converts all the characters to lower case
- `toUpperCase()`
  - Converts all the characters to upper case
- `concat(`*`anotherString`*`)`
  - Concatenates String with anotherString

http://docs.oracle.com/javase/7/docs/api/

Try it!

```
String s1 = "abcdefg";
println( s1.charAt(0) );
```

```
String s1 = "abcdefg";
String s2 = "abcdefg";
if (s1.equals(s2)) println("They are equal");
```

```
String s1 = "abcdefg";
println( s1.indexOf('c') );
```

```
String s1 = "abcdefg";
println( s1.substring(2, 5) );
```

```
println( "abcdefg".length() );
```

```
println( "abcdefg".toUpperCase() );
```

Building Strings – Use '+'

```
void setup() {
  String s1 = "Hello";
  String s2 = "World";
  String s3 = s1 + " " + s2;
  println( s3 );
}
```

```
void setup() {
  String s1 = "She is number ";
  String s2 = " in computer science.";
  String s3 = s1 + 1 + s2;
  println( s3 );
}
```

Numbers are converted to Strings prior to concatenation

Strings can be held by Arrays

- (Just like any other object or primitive type)

```
String[] tokens = new String[5];

void setup() {

  tokens[0] = "one";
  tokens[1] = "two";
  tokens[2] = "three";
  tokens[3] = "four";
  tokens[4] = "five";

  println(tokens);
}
```

```
[0] "one"
[1] "two"
[2] "three"
[3] "four"
[4] "five"
```

Strings can be held by Arrays

- Initialized when declared

```
String[] tokens = new String[] {"one", "two", "three", "four", "five"};

void setup() {
  println(tokens);
}
```

```
[0] "one"
[1] "two"
[2] "three"
[3] "four"
[4] "five"
```

Strings can be held by Arrays

- Not initialized

```
String[] tokens = new String[5];

void setup() {
  println(tokens);
}
```

```
[0] null
[1] null
[2] null
[3] null
[4] null
```

Built-in String functions (not methods)

split(*bigString, splitChar*)

- Breaks a String into a String Array, splitting on splitChar
- Returns new String Array

splitTokens(*bigString, splitCharString*)

- Breaks a String into a String Array, splitting on any char in splitCharString

join(*stringArray, joinChar*)

- Builds a new String by concatenating all Strings in stringArray, placing joinChar between each
- Inverse of split() function

text(*theString, x, y*)

text(*theString, x, y, width, height*)

- Draws *theString* on the sketch at (x, y)

Split a String based on a single or multiple separator chars

```
String s1 = "12, 34, 56";
String[] as;

void setup() {
  as = split(s1, ",");
  println( as );
}
```

```
[0] "12"
[1] " 34"
[2] " 56"
```

Creates String array
… no "new" statement

```
String s1 = "Data: 12, 34, 56";
String[] as;

void setup() {
  as = splitTokens(s1, ":,");
  println( as );
}
```

```
[0] "Data"
[1] " 12"
[2] " 34"
[3] " 56"
```

Join a String Array with a join char

```
String[] as = new String[] {"one", "two", "buckle my shoe"};

void setup() {
  String s1 = join( as, " | " );
  println( s1 );
}
```

```
one | two | buckle my shoe
```

Given the commands:

```
String aPalindrome = "a man, a plan, a canal Panama";
String[] strs = splitTokens(aPalindrome, ",");
```

Answer the following questions:

(3 pts) What will be the length of strs?

a) 1
b) 2
c) 3
d) 4

(3 pts) What will be the value of strs[1]?

a) "a man"
b) "a plan"
c) "a canal Panama"
d) 3

(3 pts) Write the expression used to obtain the number of elements in strs.

Consider the following array:

```
float[] vals = new float[]{ 1, 3, 6, 8, 9, 13, 19, 23, 32, 40 };
```

We could use the following code to determine whether the value x is in the array:

```
float x = 10;
boolean containsValue = false;
for (int i=0; i < vals.length; i++) {
    if (vals[i] == x) {  // comparison
        containsValue = true;
    }
}
```

However, in the worst case, this method requires vals.length (10) comparisons.

5.1 (10 pts) Describe *in detail* how the binary search algorithm would find whether x is in the array. Make certain to describe how the algorithm works.

5.2 (5 pts) How many comparisons would binary search take to solve the same problem? Justify your answer if you're not certain.

The following program was designed to count and print the number of duplicates in the myArray String array. Unfortunately, it doesn't work properly. When I test it with the given data, it tells me that I have 11 duplicates, but I know that there are only two. Fix the program so that it works correctly.

```
// Count and print the number of duplicate strings in myArray

String [] myArray = {"A", "B", "C", "D", "A", "F", "C"};

void setup() {
  int count = 0;

  for (int i=0; i<myArray.length; i++) {
    for (int j=0; j<myArray.length; j++) {
      if (myArray[i].equals( myArray[j] )) {
        count++;
      }
    }
  }

  println("There are " + count + " duplicates.");
}
```