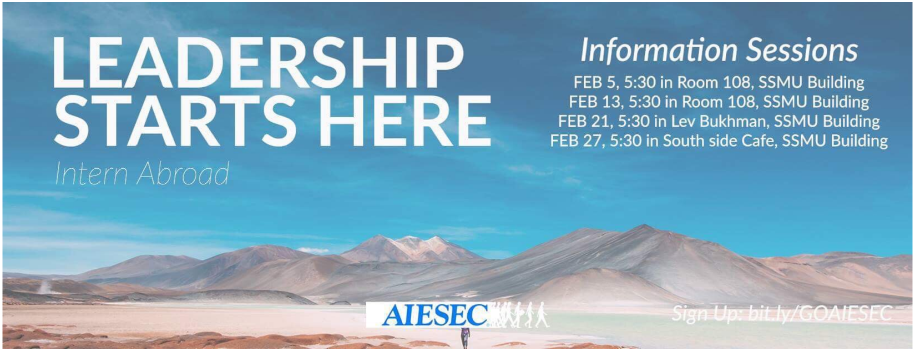
Lecture Feb 12 - Nested Loops and Arrays

Bentley James Oakes

February 11, 2018

AISEC

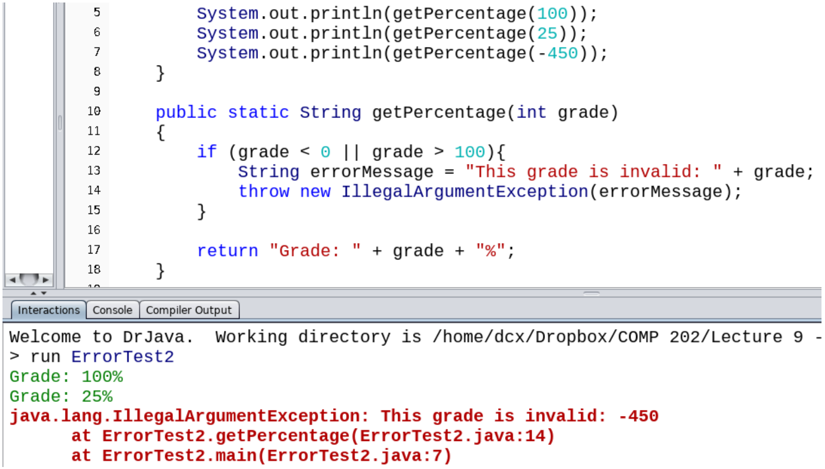

LEADERSHIP STARTS HERE
Intern Abroad
Information Sessions
FEB 5, 5:30 in Room 108, SSMU Building
FEB 13, 5:30 in Room 108, SSMU Building
FEB 21, 5:30 in Lev Bukhman, SSMU Building
FEB 27, 5:30 in South side Cafe, SSMU Building
AIESEC
Sign Up: bit.ly/GOAIESEC

This Lecture

Section 1

Throwing Exceptions

Error Checking



```
        System.out.println(getPercentage(100));
        System.out.println(getPercentage(25));
        System.out.println(getPercentage(-450));
    }

    public static String getPercentage(int grade)
    {
        if (grade < 0 || grade > 100){
            String errorMessage = "This grade is invalid: " + grade;
            throw new IllegalArgumentException(errorMessage);
        }

        return "Grade: " + grade + "%";
    }
```

```
Welcome to DrJava.  Working directory is /home/dcx/Dropbox/COMP 202/Lecture 9 -
> run ErrorTest2
Grade: 100%
Grade: 25%
java.lang.IllegalArgumentException: This grade is invalid: -450
	at ErrorTest2.getPercentage(ErrorTest2.java:14)
	at ErrorTest2.main(ErrorTest2.java:7)
```

- We specify an Exception to throw
- And providing a message (optional but recommended)
- `throw new IllegalArgumentException(message)`

Section 2

Break and Continue

Break and Continue

- This section has two constructs to help you design loops
- These are the **break** and **continue** constructs
- **These don't have to be used**
- Can help or hurt the readability of your code

Break

```
String s = "This is a sentence. Another one.";
boolean keepRunning = true;
for (int i = 0; i < s.length() && keepRunning; i++){
    char c = s.charAt(i);
    System.out.print("|" + c + "|");

    //stop the loop if the character
    //is a period
    if (c == '.'){
        keepRunning = false;
    }
}
System.out.println();
```

```
|T||h||i||s|| ||i||s|| ||a|| ||s||e||n||t||e||n||c||e||.|
```

- This example stops the loop when a period is reached

Break

- The *break* statement stops the loop immediately
- This code does the same thing as the last slide

```java
String s2 = "This is a sentence. Another one.";
//loop through the sentence
for (int i = 0; i < s2.length(); i++){
    char c = s2.charAt(i);
    System.out.print("|" + c + "|");

    //stop the loop if the character
    //is a period
    if (c == '.'){
        break;
    }
}
System.out.println();
```

Continue

- The *continue* statement moves to the next iteration of the loop
- This example skips printing spaces

```
String r = "Hello this is a sentence with words";

//loop through the sentence
for (int i = 0; i < r.length(); i++)
{
    //get each character
    char c = r.charAt(i);

    //skip this character if it is a space
    //this will go to the next iteration
    //of the loop
    if (c == ' '){
        continue;
    }

    //print the character
    System.out.print(c);
}
```

```
Hellothisisasentencewithwords
```

Comparison

Both `break` and `continue` can only be used within a loop

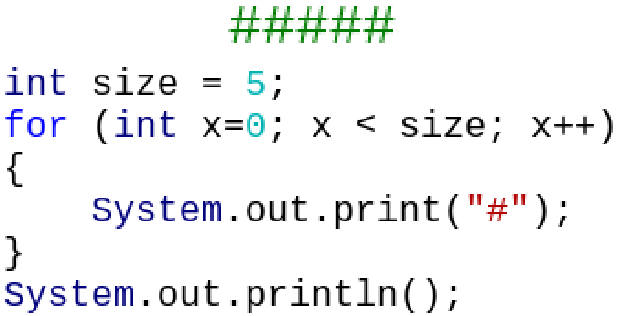
Break	Stops the loop immediately
Continue	Skips the rest of the instructions in the iteration

Section 3

Nested For Loops

Nested For Loops

- Let's start with printing a line

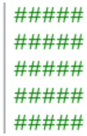


```
#####
```

```
int size = 5;
for (int x=0; x < size; x++)
{
    System.out.print("#");
}
System.out.println();
```

Nested For Loops

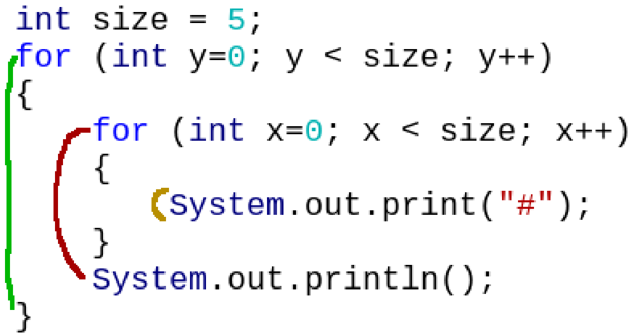
- Now let's repeat the line a few times



```
#####
#####
#####
#####
#####
```

```
int size = 5;
for (int y=0; y < size; y++)
{
    for (int x=0; x < size; x++)
    {
        System.out.print("#");
    }
    System.out.println();
}
```

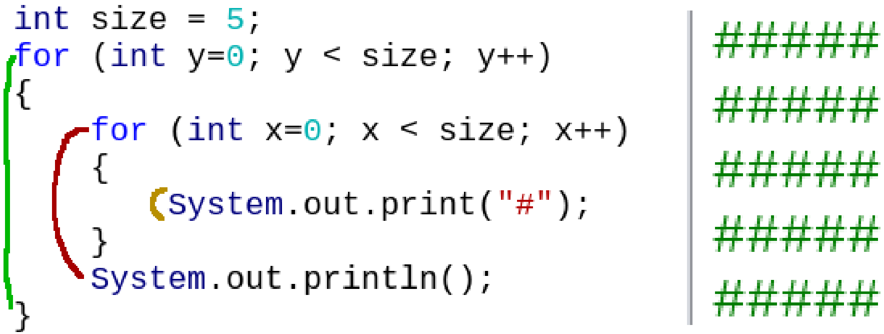
Nested For Loops



```
int size = 5;
for (int y=0; y < size; y++)
{
    for (int x=0; x < size; x++)
    {
        System.out.print("#");
    }
    System.out.println();
}
```

- We are just repeating the line printing five times
- Note that the *inner for loop* is restarted five times

Nested For Loops



```
int size = 5;
for (int y=0; y < size; y++)
{
    for (int x=0; x < size; x++)
    {
        System.out.print("#");
    }
    System.out.println();
}
```

```
######
######
######
######
######
```

- The *outer for loop* has five iterations
- The *inner for loop* has five iterations for every iteration of the *outer for loop*
- So the `System.out.print` statement executes 25 times

Drawing a Triangle

- Let's make the *inner for loop* depend on the *outer for loop*
- We will make this shape:

```
#
##
###
####
#####
```

Triangle Drawing Code

- We change the condition in the inner for loop

```
int size = 5;
for (int y=0; y < size; y++)
{
    //make the inner for loop depend on y
    //note the condition 'x < y + 1'
    for (int x=0; x < y + 1; x++)
    {
        System.out.print("#");
    }
    System.out.println();
}
```

```
#
##
###
####
#####
```

- Now the inner for loop will stop at different times, depending on the value of *y*

Triangle Drawing Code

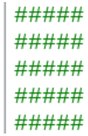
- To help understanding,
- We can print out the value of x instead

```
int size = 5;
for (int y=0; y < size; y++)
{
    //make the inner for loop depend on y
    //note the condition 'x < y + 1'
    for (int x=0; x < y + 1; x++)
    {
        //print out the value of x
        System.out.print(x);
    }
    System.out.println();
}
```

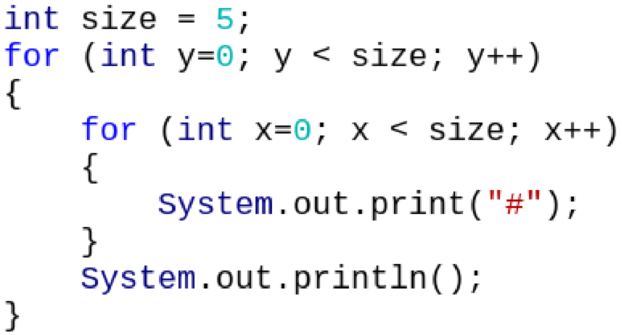
```
0
01
012
0123
01234
```

Hash Symbol

- Let's rewind a bit, and go back to just a box



```
#####
#####
#####
#####
#####
```



```java
int size = 5;
for (int y=0; y < size; y++)
{
    for (int x=0; x < size; x++)
    {
        System.out.print("#");
    }
    System.out.println();
}
```

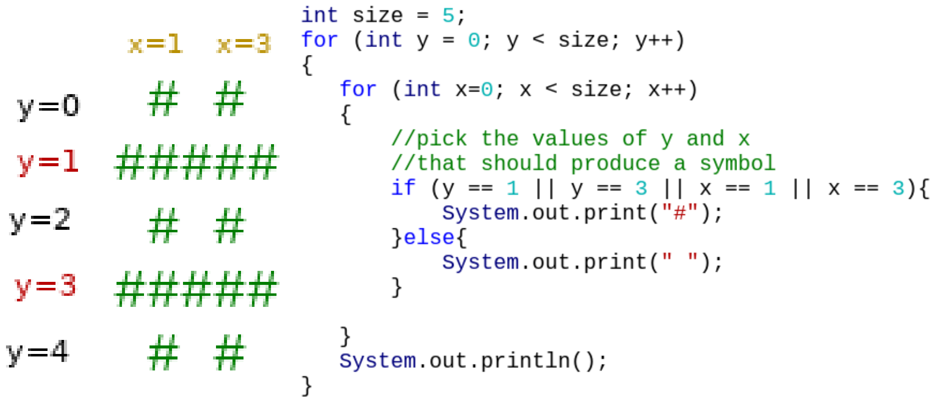
- We're going to add some *if statements* to make a hash symbol #
  - Not called a *hashtag*!

Hash Symbol

```
 # #
#####
 # #
#####
 # #
```

```
int size = 5;
for (int y = 0; y < size; y++)
{
   for (int x=0; x < size; x++)
   {
       //pick the values of y and x
       //that should produce a symbol
       if (y == 1 || y == 3 || x == 1 || x == 3){
           System.out.print("#");
       }else{
           System.out.print(" ");
       }

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

Coordinate System



```
 # #
#####
 # #
#####
 # #
```

```java
int size = 5;
for (int y = 0; y < size; y++)
{
   for (int x=0; x < size; x++)
   {
       //pick the values of y and x
       //that should produce a symbol
       if (y == 1 || y == 3 || x == 1 || x == 3){
           System.out.print("#");
       }else{
           System.out.print(" ");
       }

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

- This defines a coordinate system
- Prints different symbols based on the x and y coordinates

Section 4

Arrays

Motivation

- Now that we have *for-loops*, we can start iterating values in a collection
- For example, what if we wanted to get the average/minimum/maximum temperature for February in Montreal?

Weather

```
//the average temp for the first
//three days of February
double feb1 = -4.2;
double feb2 = -1.9;
double feb3 = -146;
//...Enter in all the days here...

double avg = (feb1 + feb2 + feb3)/3;
System.out.println("Average: " + avg);
```

- If we create variables for every day in February, our code will be very tedious to write

Arrays

- We'll create a list of values, all with the same type
- As well, the list is ordered - we can talk about the first position in the list, the second position, and so on

Example: A shopping list

- 0. Naan bread
- 1. Cheese curds
- 2. Ice cream
- 3. Smoked meat
- 4. Chocolate bars
- 5. Hot sauce

Creating an Array

When we make a array, it's like creating a list of something
Two things to think about:

- How long is my array?
- What type am I storing in this array?

Declaring an Array

- Just like other variables, we need to declare our *array variable*

Examples:

- `int[] grades;`
- `double[] febTemps;`
- `String[] catNames;`
- `String[] args;`

- You've seen this last one
- Note the square brackets. They mean that this variable is an array that stores elements of that type

Creating an Array

- Let's create an array and fill it with values

```
String[] threeMonths = new String[3];
threeMonths[0] = "January";
threeMonths[1] = "February";
threeMonths[2] = "March";
```

- Arrays are created with the `new` keyword
- `threeMonths` will be created with a length of three
  - The length of an array can't be changed after it's created
- The entries are then filled after array creation

Accessing the Array

- As seen above, the square brackets allow us to access elements in the array

Examples:

- Change element in the array: `grades[1] = 100;`
    - Assign to the second position in the array (index 1) the value 100
- Read value from the array: `int x = grades[5];`
    - Read the value at the sixth position (index 5) from the array, and assign it to a variable
- Combination: `grades[3] = grades[5];`
    - Read the value from index 5, and place it in index 3

Iterating an Array

- Let's use a *for-loop* to iterate through an array

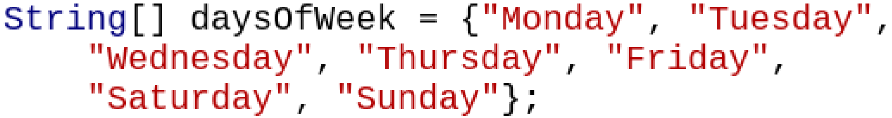
```
String[] catNames = {"Jack Bauer", "Lord Fuzzykins", "Mrs. Whiskers"};
System.out.println("Length: " + catNames.length); //prints 3

for(int i=0; i < catNames.length; i++)
{
    System.out.println(catNames[i]);
}
```

Notice the difference:

- For String s the length is `s.length()`
- For arrays, the length is `catNames.length`

Creating an Array

- Let's declare and initialize an array in one step



```
String[] daysOfWeek = {"Monday", "Tuesday",
    "Wednesday", "Thursday", "Friday",
    "Saturday", "Sunday"};
```

- `daysOfWeek` will have a length of seven
- Index 0 will contain "Monday"
- Index 6 will contain "Sunday"

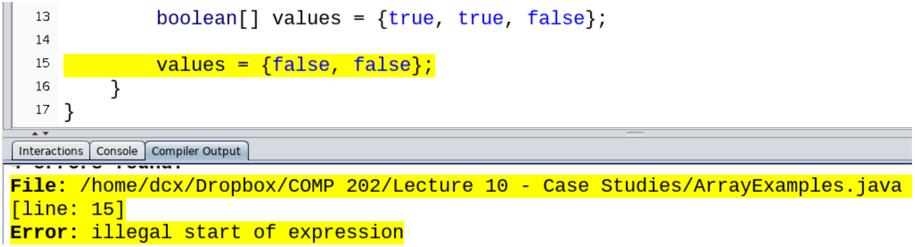
Creating an Array Examples

- More examples:

```
String[] names = { "Bentley", "Giulia", "Batman"};

int[] numbers = {3, 4, 6, 0, 10};

double[] dec = {45.3, 232.4};

boolean[] values = {true, true, false};
```

- We use braces to set the initial value of the array
- **This technique can only be used when creating a new array!**

Creating an Array Examples



```
        boolean[] values = {true, true, false};

        values = {false, false};
    }
}
```

```
File: /home/dcx/Dropbox/COMP 202/Lecture 10 - Case Studies/ArrayExamples.java
[line: 15]
Error: illegal start of expression
```

- To repeat, we can't use the braces to assign values to an array after it is created

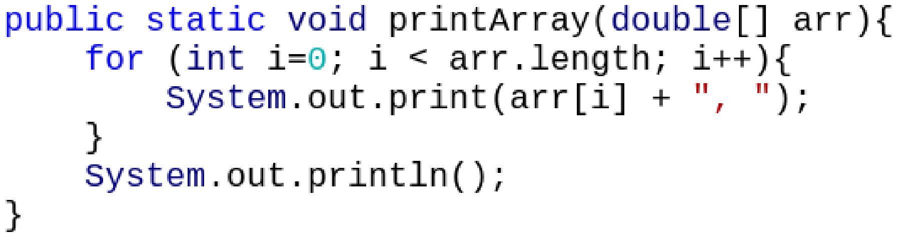
Array Example

- Let's creates an array of doubles and then increase the value of each element in the array by five

```
//create an array of doubles
double[] values = {5.67, 2.1, 4.5, 99};

//add 5 to each value
for (int i=0; i < values.length; i++){
    values[i] = values[i] + 5;
}

//prints 10.67, 7.1, 9.5, 104.0
for (int j=0; j < values.length; j++){
    double dbl = values[j];
    System.out.print(dbl + ", ");
}
```

Array Printing

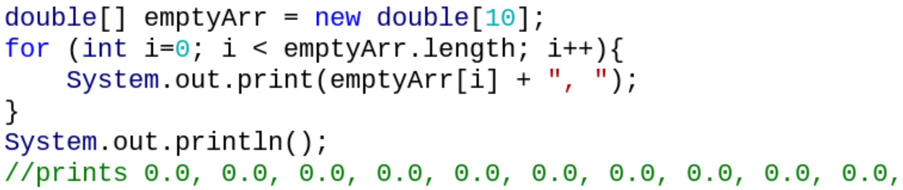
- It's very helpful to have a method just for printing out an array



```
public static void printArray(double[] arr){
    for (int i=0; i < arr.length; i++){
        System.out.print(arr[i] + ", ");
    }
    System.out.println();
}
```

- Note that this method can only accept `double` arrays as a parameter

Printing After Creation



```java
double[] emptyArr = new double[10];
for (int i=0; i < emptyArr.length; i++){
    System.out.print(emptyArr[i] + ", ");
}
System.out.println();
//prints 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
```

- Note that this does work
- What's inside an array if we try to print before we assign values?

Default Values

When you create an array, Java will assign default values to each position. The values are:

- For int/double arrays: 0
- For boolean arrays: false
- For char arrays: a special value
- For String arrays: a special value **null**
  - We'll see more about **null** later
  - It's a placeholder to say "there's no value here"

Null

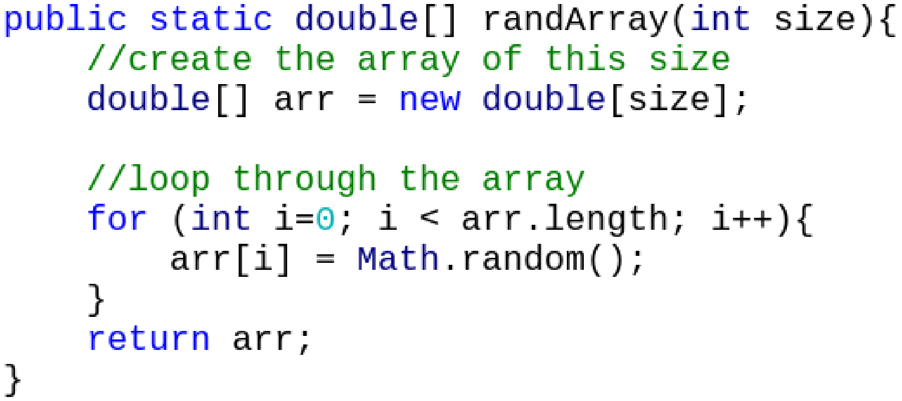
```
String[] sArr = new String[5];
for (int i=0; i < sArr.length; i++){
    System.out.print(sArr[i] + ", ");
}
//prints null, null, null, null, null,
```

Section 5

Array Examples

Random Array

- Let's write a method with a parameter `n`
- This method will return an array of size `n`
- And each position will be filled with a random number between 0 and 1



```java
public static double[] randArray(int size){
    //create the array of this size
    double[] arr = new double[size];

    //loop through the array
    for (int i=0; i < arr.length; i++){
        arr[i] = Math.random();
    }
    return arr;
}
```

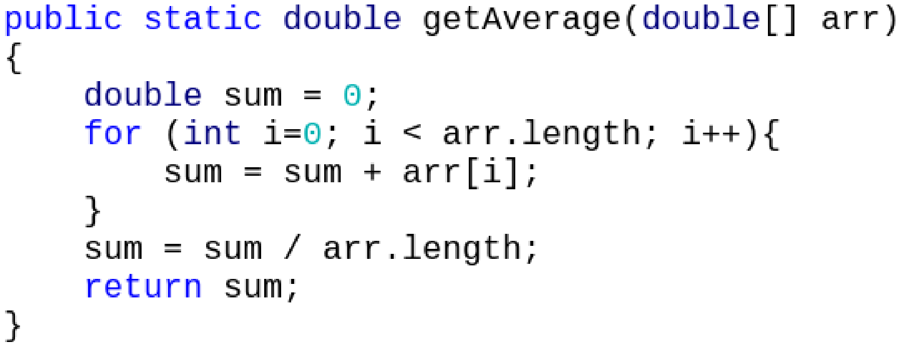
Get Random Entry

- Let's write a method which accepts a `String` array
- It will return a random entry from the array

```
public static void main(String[] args){
    String[] planets = {"Mercury", "Venus", "Earth", "Mars",
    "Jupiter", "Saturn", "Uranus", "Neptune"};

    String randPlanet = randArray(planets);
    System.out.println("I want to visit: " + randPlanet);
}

public static String randArray(String[] arr){
    //create a random number from 0 to arr.length - 1
    int rand = (int) (Math.random() * arr.length);

    //return that entry
    return arr[rand];
}
```

Finding the Average of an Array

- Let's find the average of an array



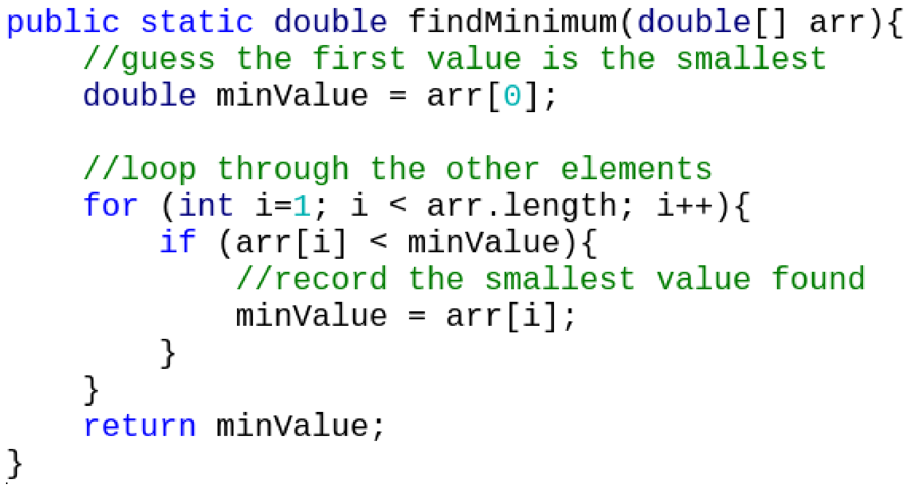
```java
public static double getAverage(double[] arr)
{
    double sum = 0;
    for (int i=0; i < arr.length; i++){
        sum = sum + arr[i];
    }
    sum = sum / arr.length;
    return sum;
}
```

- Be on the lookout for integer division here!

Finding the Minimum

- Finding the minimum of an array is similar to code we've seen before
- Keep track of the smallest element seen so far
- Start off by assuming that the first element of the array is the smallest
- And then check the others

Finding the Minimum



```java
public static double findMinimum(double[] arr){
    //guess the first value is the smallest
    double minValue = arr[0];

    //loop through the other elements
    for (int i=1; i < arr.length; i++){
        if (arr[i] < minValue){
            //record the smallest value found
            minValue = arr[i];
        }
    }
    return minValue;
}
```

- Note that this method will crash if the array is empty
  - With an `ArrayIndexOutOfBoundsException`

Reverse Array

Write a method that takes as input an array and does not return anything. This method should reverse the order of the elements in the array.

Reverse Array

```
public static int[] reverse(int[] arr){
    //create new array of same size
    int[] newArr = new int[arr.length];

    for (int i=0; i < newArr.length; i++){

        //get the position in the old array
        int oldIndex = arr.length - 1 - i;

        //store the value
        newArr[i] = arr[oldIndex];
    }
    return newArr;
}
```

Section 6

Primitive versus Reference Types Intro

Type Division

- Arrays are different than the usual variable types `int`, `double`, `boolean`, `char`
- Arrays are more complicated, like `Strings`

We separate variable types in Java into two groups:

- Primitive types
- Reference types

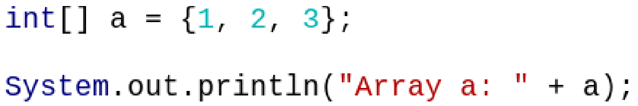
Primitive vs Reference Types

- Primitive types:
  - `int, double, boolean, char`
- Reference types:
  - `String`
  - Arrays
  - Objects

What's different about reference types?

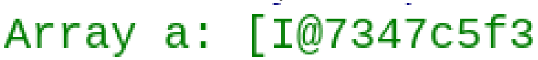
- Can't use == for comparisons
- Variables store **addresses** instead of values
- We can call methods and access members of variables
  - `.equals()` for Strings, `.length` for arrays

Reference Type Variable



```
int[] a = {1, 2, 3};

System.out.println("Array a: " + a);
```

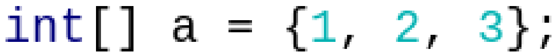
What prints?



```
Array a: [I@7347c5f3
```

This is the **address** of the data within a

Addresses in Variables



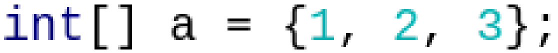
```
int[] a = {1, 2, 3};
```

- The value stored in a is the address in the computer's memory where we can find those numbers

Analogy:

- A reference variable stores the website address where the data can be found

Addresses Figure



```
int[] a = {1, 2, 3};
```

Address	Variable Type	ID	Value
1171...	`int[]`	`a`	@7347...
...			
7347...	`int`	`a[0]`	1
7347...	`int`	`a[1]`	2
7347...	`int`	`a[2]`	3

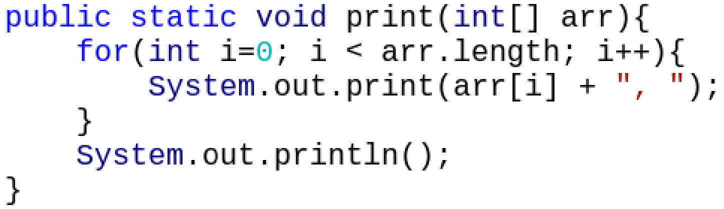
- The array only stores the address where the data starts
- When we index, we are looking up the address in the computer's memory

Consequences

- **Reference type variables store addresses**
- This means that printing out their value might not work
- Also means that we have to compare them a different way
  - This is why we can't use == to compare Strings
- Another consequence: we can have two variables storing the same address

Printing an Array

Let's make sure we know how to properly print an array



```
public static void print(int[] arr){
    for(int i=0; i < arr.length; i++){
        System.out.print(arr[i] + ", ");
    }
    System.out.println();
}
```

Aliasing

```
//create a
int[] a = {1, 2, 3};
print(a);

//assign the address in a
//into b
int[] b = a;
print(b);

//change the first element in b
b[0] = 5;

//print out a again
print(a);
```

What prints?

1, 2, 3

1, 2, 3

5, 2, 3

- Here we have two array variables pointing to the **same address**
- A change in one affects the other
- This is called **aliasing**
- Analogy: They both contain the same website address, so changes are seen for both

Conclusion

- **Primitive types store values**
  - `int, double, boolean, char`
- **Reference types store addresses**
  - `Strings`, arrays, Objects
- We'll examine consequences for comparisons, aliasing, and swapping

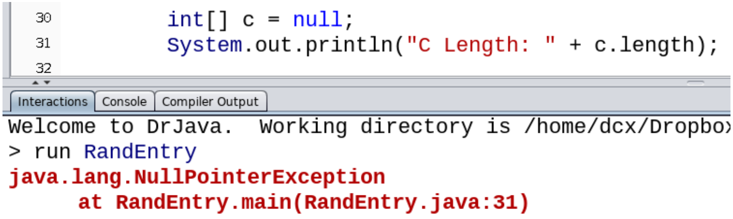
Section 7

Null

null

- Reference type variables can also store the **null** value.
- `null` means *no address*
  - Analogy: The website address is a big red X
- Null is useful to check if something has not been initialized yet
- We'll see examples of using `null` later

NullPointerException



```
        int[] c = null;
        System.out.println("C Length: " + c.length);

```

```
Welcome to DrJava.  Working directory is /home/dcx/Dropbo
> run RandEntry
java.lang.NullPointerException
      at RandEntry.main(RandEntry.java:31)
```

- This is a common run-time error
- Occurs when a reference variable has the value *null* and you try to access it
    - Example: Trying to access `c.length` if c is `null`
    - Or trying to print out the first element in c

Section 8

Equality and Imports

Comparing Reference Types

- Now we've seen that reference type variables store addresses
- Let's talk about comparing reference type variables

Comparing Strings

- We talked about comparing `Strings` with `.equals()`, instead of the == we use for primitive types
- This is because a `String` variable is actually storing an address
- And == is comparing the addresses

- You can print out a `String` directly because Java does some work behind the scenes

Comparing Two Arrays

Write a method that takes as input two `integer` arrays and tests whether they contain the same elements

```
int[] a = {1, 2, 3};
int[] b = {1, 2, 3};

boolean areSame = (a==b);
System.out.println("Are same: " + areSame);
//Are same: false
```

Comparing Arrays

- Let's try to use `.equals()`

```
int[] a = {1, 2, 3};
int[] b = {1, 2, 3};

boolean areSame = (a==b);
System.out.println("Are same: " + areSame);
//Are same: false

boolean areSameEquals = a.equals(b);
System.out.println("Are same using equals: " + areSameEquals);
//Are same using equals: false
```

This prints *false* again
This worked with `Strings`!

Comparing Arrays

There are two ways to compare arrays:

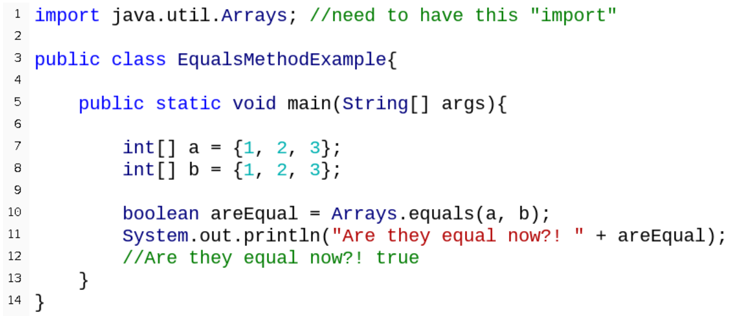
- Write your own method
  - I highly recommend you do this as practice
- Use the `Arrays.equals()` method

Array.equals()

```java
import java.util.Arrays; //need to have this "import"

public class EqualsMethodExample{

    public static void main(String[] args){

        int[] a = {1, 2, 3};
        int[] b = {1, 2, 3};

        boolean areEqual = Arrays.equals(a, b);
        System.out.println("Are they equal now?! " + areEqual);
        //Are they equal now?! true
    }
}
```

- This will compare the contents of two arrays and return a boolean value
- At the top of our .java file, we have to add `import java.util.Arrays;`

Import Statements



```java
import java.util.Arrays; //need to have this "import"

public class EqualsMethodExample{

    public static void main(String[] args){

        int[] a = {1, 2, 3};
        int[] b = {1, 2, 3};

        boolean areEqual = Arrays.equals(a, b);
        System.out.println("Are they equal now?! " + areEqual);
        //Are they equal now?! true
    }
}
```

- The top line is an example of an **import statement**
- Required to use methods in the Arrays class
- All your import statements go at the top of your .java file, before your `public class`

Other Arrays Methods

Here are some useful methods in the Arrays class:

- `Arrays.equals(a,b)` $\rightarrow$ returns a boolean indicating whether or not the contents of the two arrays are the same.
- `Arrays.toString(arr)` $\rightarrow$ returns the contents of an array as a `String` value. This allows us to print the contents of an array without using a *for-loop*.
- `Arrays.sort(arr)` $\rightarrow$ sorts the input array in increasing order

- Note: It would be a **perfect test question** to ask you to write the `.equals()` and `toString()` methods by hand
- Writing the `sort()` method is more difficult, and won't be testable material, but it is also great practice