Objectives

- Last of Java fundamentals
  - Arrays wrap up
  - Indefinite loops
  - For-each loops
  - Switch statements
- Static methods and fields

1

Review

- What is the difference between ***declaring***, ***initializing,*** and ***defining*** a variable?
- What is the syntax of a `for` loop?
- What is the scope of a variable?
- What are the relational operators (and, or, not) in Java?
- How do we access command-line arguments from a Java program execution?
- What does `static` mean?
  - How do we make a *constant* for a *class*?
- Arrays:
  - How do we declare an array of elements?
  - How do we access elements of an array?
  - How can we find out the size of an array?

2

Terminology Review

- Declaration: `int x;`
- Definition: `x = 3;`
- Initializing: typically the first time the variable is given a value
  - `int x = 3;`
  - Or could be first assignment, e.g., `x = 3;`

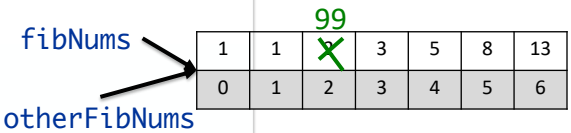
Review: Arrays

- Assigning one array variable to another ➔both variables refer to the same array
  - Similar to Python
- Draw picture of below code:

```
int [] fibNums = {1, 1, 2, 3, 5, 8, 13};
int [] otherFibNums;

otherFibNums = fibNums;
otherFibNums[2] = 99;

System.out.println(otherFibNums[2]);
System.out.println(fibNums[2]);
```



Displays:
99
99

Assign 2 Discussion

- Java Conventions:
  - Class names begin with capital letter
  - Variable names begin with lowercase letter
  - Class constants: name with all capital letters, e.g., `DIFFICULTY_SCORE`
- Can fully-specify the class instead of importing class, e.g.,

```
java.util.Arrays.sort(myArray);
```


Danger of a Large Library

- Be careful when searching for classes
- Lots of classes seem like they're what we want but aren't, e.g.,

```
java.lang.reflect.Array
javax.sql.rowset.serial.Array
```

An array (e.g., `int[] array`) is **not** an instance of a class so we cannot call methods on it.

Review: Array Length

- All array variables have a ***field*** called `length`
  - Note: no parentheses because not a method

```
int[] array = new int[10];
for (int i = 0; i < array.length; i++) {
     array[i] = i * 2;
}

for (int i = array.length-1; i >= 0; i--) {
     System.out.println(array[i]);
}
```

I'm declaring `i` twice in this code. Why is that not a compiler error?

Review: Variable Scope

- All array variables have a ***field*** called `length`
  - Note: no parentheses because not a method

```
int[] array = new int[10];
for (int i = 0; i < array.length; i++) {
     array[i] = i * 2;
}

for (int i = array.length-1; i >= 0; i--) {
     System.out.println(array[i]);
}
```

Scope of i is within each for loop; other loop doesn't see it.
Declaring counter variable in for loop is common practice.

MORE CONTROL STRUCTURES

Control Flow: foreach Loop

- Introduced in Java 5
  - Sun called "enhanced for" loop
- Iterate over all elements in an array (or Collection)
  - Similar to Python's for loop

```
int[] a;
int result = 0;
. . .        "in"
for (int i : a)
{
   result += i;
}
```

https://docs.oracle.com/javase/8/docs/technotes/guides/language/foreach.html

for each int element i *in* the array a , the loop body is executed

Control Flow: `while` Loops

- `while` loop
  - Condition must be enclosed in parentheses
  - Body of loop must be enclosed in `{}` if multiple statements

```
int counter = 0;
while (counter < 5) {
      System.out.println(counter);
      counter++;
}
System.out.println("Done: " + counter);
```

shortcut

Changing control flow: `break`

- Exits the current loop

```
while ( <readingdata> ) {
   …
   if( <something> ) { // now we're done!
     break;
   }
}
```

Control Flow: `switch` statement

- Like a big `if/else if` statement
- Works with variables with datatypes `byte`, `short`, `char`, `int`, and `String`

```
int x = 3;
switch(x) {
      case 1:
            System.out.println("It's a 1.");
            break;
      case 2:
            System.out.println("It's a 2.");
            break;
      default:
            System.out.println("Not a 1 or 2.");
}
```

13

Control Flow: `switch` statement

```
switch(grade) {
      case 'a':
      case 'A':
            System.out.println("Congrats!");
            break;
      case 'b':
      case 'B':
            System.out.println("Not too shabby!");
            break;
      … // Handle c, d, and f …
      default:
            System.out.println("Error: not a grade");
}
```

 Grades.java

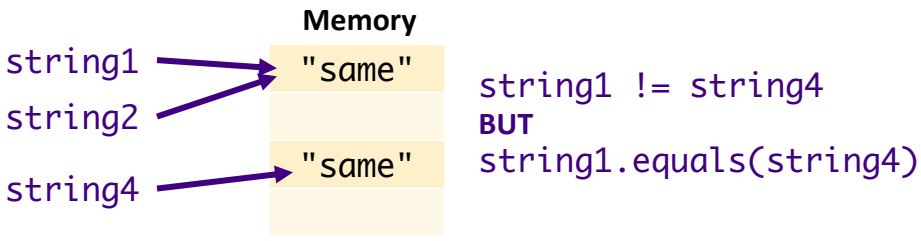
14

Python Gotcha: String Comparisons

- `string1 == string4` will **not** yield the same result as `string1.equals(string4)`
  - == tests if the *objects* are the same
    - **not** if the *contents* of the objects are the same
  - Similar to `is` operator in Python

Python Gotcha: String Comparisons

- `string1 == string4` will **not** yield the same result as `string1.equals(string4)`
  - == tests if the *objects* are the same
    - **not** if the *contents* of the objects are the same
  - Similar to `is` operator in Python



Summary: Python to Java Gotchas

- Every variable needs to be declared before it is used
- Every variable needs a statically declared data type
- Scope of variables
- Need to use `equals` method to do more than just a "same object" check
- Syntax
  - Semicolons at the end of **statements**
  - Braces around blocks of code
  - Keywords

Benefits of Static Typing

- Easier to remember type of variable
  - Know operations that can be executed on a variable of a certain type
- Compiler can check that you're only using valid operations for this type
- More benefits later this semester

STATIC METHODS AND FIELDS

Static Methods/Fields

- For functionality/data that is specific to a *class*
  - And is **not** specific to a particular object
- `java.lang.Math`
  - No constructor (what does that mean?)
  - Static fields: `PI`, `E`
  - Static methods:
    - `static double sin(double a)`

Static Methods

- Do **not** operate on objects
  - Can**not** access *instance* fields of their class
- Can access *static fields* of their class
  - Example: `Math` class could have a static method that uses `PI`
- Similar to Python *functions* that are associated with the class

Analyzing `java.lang.String` API

- `String toUpperCase()`
  - Converts all of the characters in this String to upper case
- `static String valueOf(boolean b)`
  - Returns the string representation of the `boolean` argument

Why can/should the second method be `static`?

Discussion

Why is `main` static?

`main()`

- Most common static method
- `main()` does not operate on any objects
  - Runs when a program starts
  - There are no objects yet
- `main()` executes and constructs the objects the program needs and will use
  - Like the ***driver function*** for the program

JavaDocs for Methods

```
/**
 * Returns the string representation of the boolean argument.
 *
 * @param b – a boolean
 * @return if the argument is true, a string equal to "true" is
 *         returned; otherwise, a string equal to "false" is
 *         returned.
 */
public static boolean valueOf(boolean b) {
```

- Use format similar to class comments
- Use `@param` tag(s) to describe what method takes as parameter(s)
- Use `@return` tag to describe what method returns

JavaDocs for Methods

```
/**
 * Returns the string representation of the boolean argument.
 *
 * @param b – a boolean
 * @return if the argument is true, a string equal to "true" is
 *         returned; otherwise, a string equal to "false" is
 *         returned.
 */
public static boolean valueOf(boolean b) {
```

Generated on Web Page: https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/lang/String.html#valueOf(boolean)

valueOf

```
public static String valueOf(boolean b)
```

Returns the string representation of the boolean argument.

Parameters:

b - a boolean.

Returns:

if the argument is true, a string equal to "true" is returned; otherwise, a string equal to "false" is returned.

Static Summary

- Static fields and methods are **part of a class** and **not** an object
  - Do not require an object of their class to be created to use them
- When would we make a method `static`?
  - When a method does not have to access an object's state (fields) because all needed data are passed into the method
  - When a method only needs to access static fields in the class

Practice

- Create a new `static` method called `average` that
  - Takes as parameters 2 integers
  - Returns the average of those 2 numbers
- Why should this be a static method?
- What should the signature of this method look like?
  - Use the `main` method's signature to guide you

Making a Program

- `main` will prompt user for two integers and display the result of getting the average
  - `main` is the driver
- `main` should either be the first or the last method in the class
  - i.e., not somewhere in the middle

- How do we call the method?

Looking Ahead

- Assignment 3 – due Friday
- Textbook: Read through Loops and Iteration