Lecture Feb 26 - Intro to Objects

Bentley James Oakes

February 26, 2018

No Office Hours Today

- No office hours today (Feb 26th)
- Will have longer office hours on Wednesday

This Lecture

1. Midterm
2. Substring
3. 2D Arrays
4. Printing 2D Arrays
5. 2D Array Examples
6. Intro To Objects
7. Random Class

Section 1

Midterm

Midterm

- Midterm on **March 13th, 18:00 to 21:00**
- If you miss the midterm, your final exam will be worth 65% of your final grade
- Format:
  - True/False
  - Short Answer
  - Long Answer (multiple parts)
- Every topic seen so far in the course
  - Does not include objects

Midterm

- Closed book examination
- A legal-sized (8.5" by 14") crib sheet is permitted
  - Single or double-sided, handwritten or typed
- Non-electronic translation dictionaries are permitted
- Otherwise, only writing implements (pens, **pencils**, erasers, pencil sharpeners, etc.) are allowed
- Possession of any other tools or devices is prohibited.

Midterm

- Documentation of `String`, `print`, and `Math` methods are provided

`String` (package `java.lang`) Methods:

- `public boolean equals(Object anObject)`: Compares this `String` to `anObject`.
- `public int length()`: Calculates the length of this `String`.
- `public char charAt(int i)`: Gets the `char` at position `i` of the `String`. Note that counting starts from 0 so that to get the first character of the `String` you should input `i` equals 0.
- `public boolean equalsIgnoreCase(String anotherString)`: Compares, ignoring case considerations, this `String` to `anotherString`.

Material

This is not an exhaustive list!

- Binary numbers
  - Decimal-to-binary, binary-to-decimal
- Variables and primitive data types
  - Creating variables of different types
- Expressions and assignments
  - What is 'evaluation' of an expression?

Material

- Input arguments
  - What are they, and how are they used?
- Methods
  - Passing parameters, returning values
  - Calling a method, void methods
- Mod operator (%)
  - What is it? What have we used it for?
- If/else-if/else
  - What's the order they are evaluated in?

Material

- Strings and chars
  - How do we get each char in a `String`?
  - How can we test `chars`?
  - How does concatenation work?
- For and while loops
  - Loop conditions
  - Initialization/condition/modification
  - Know how many times a loop is iterated

Material

- Type Issues
  - Casting, conversions, integer division
- Different kinds of errors
  - Logic/style/compile-time/run-time
  - Examples of when each occur

Material

- Random numbers
  - How to generate a number from *min* to *max*
- Arrays
  - What do they store? How do we create them?
  - How do we do calculations on arrays?
  - How do we access elements in an array?
  - Two-dimensional arrays are testable
- Reference Types
  - What do we copy over when we call a method? Values vs addresses
  - Can we change an array in a method? A `String`?
  - What is the *null* value used for?

How to Study

- Read and re-read the slides and your notes
  - Do this until everything makes sense
- Watch the videos on MyCourses
- Do the warm-up questions on the assignments
- Do `codingbat.com` questions
  - Practice writing them out by hand
- Practice old midterms
  - Up on MyCourses now
- Look at MyCourses for midterm review sessions

Review Session

- Send me your questions
- And I'll do a short review on the day before the midterm
- (But don't wait till then to start studying...)

Advice

- Research shows that you learn the best when sleeping
- Your brains stores away the information in the proper spot
- So the best way to study is to **do it early** and **do it repeatedly**

Section 2

Substring

substring

We didn't talk about a `String` method that could be useful for CodingBat and the assignment

`public String substring(int start, int finish)`: Returns a new String composed of the `this` String starting from index `start` and up to, but not including index of `finish`

```
String test = "I like apples.";

String sub = test.substring(2, 6);

System.out.println(sub); //like
```

substring

You may be asked to write the `substring` method yourself

```
public static String substring(String s, int left, int right)
{
    //change ranges to be within 0 and s.length()
    if (left < 0){
        left = 0;
    }
    if (right > s.length()){
        right = s.length();
    }
    //now build up the result String
    String result = "";
    for (int i=left ; i < right; i++)
    {
        result += s.charAt(i);
    }
    return result;
}
```

Section 3

2D Arrays

Arrays of Arrays

Three main ways to create arrays of arrays:

1. `int[][] a = {{1, 2, 3}, {5, 6}}`
2. `int[][] b = new int[4][5];`
3. `int[][] c = new int[3][];`

Hard-coding Values

```
int[][] a = {{1, 2, 3}, {5, 6}}
```

- Creates an array of length 2
- The first element in the array is another array of length 3 (with values 1,2,3).
- The second element is an array of length 2 (with values 5,6).

Creating An Empty Array of Arrays

```
int[][] grid = new int[2][3];
```

- Creates an array of length 2, where each element in that array is another array of length 3
- Because the array store `integers`, every entry will start off with a value of 0

The array will contain `[ [0, 0, 0], [0, 0, 0] ]`

Creating an Empty Array of Arrays

```
int[][] arr = new int[2][];
```

- This creates an array of length 2.
- Each element in the array will be an array.
- However, what is currently stored is the value *null*
  - Represents an uncreated array

The array will contain `[ null, null ]`

Section 4

Printing 2D Arrays

Import Statements

```java
import java.util.Arrays;

public class MultiDimInput{
    public static void main(String[] args){
        int[][] a = {{1, 5, 6}, {3,4, 10}};

        System.out.println(Arrays.toString(a));
        //prints [[I@71ba4236, [I@153b2096]

        System.out.println(Arrays.deepToString(a));
        //prints [[1, 5, 6], [3, 4, 10]]
    }
}
```

- `Arrays.deepToString(arr)` $\rightarrow$ prints all of the elements of `arr`, where `arr` is a multi-dimensional array

As before, you may be asked to write the code for this method on an exam

Deep Printing

```
double[][] temps = {
    {-10, -20, -15, -16},
    {-20, -23, 0, -3},
    {5, -5, 2, -10}
};

for (int monthIndex = 0; monthIndex < temps.length; monthIndex++){
    double[] monthArr = temps[monthIndex];

    for (int dayIndex = 0; dayIndex < monthArr.length; dayIndex++){
        System.out.print(monthArr[dayIndex] + ", ");
    }
    System.out.println();
}
//prints
//-10.0, -20.0, -15.0, -16.0,
//-20.0, -23.0, 0.0, -3.0,
//5.0, -5.0, 2.0, -10.0,
```

- Outer for-loop for months, inner for-loop for days in the months

Null in Arrays

```
int[][] a = {
    {1, 2, 5}, null
};
System.out.println(Arrays.deepToString(a));
//prints [[1, 2, 5], null]
```

- Note that it's possible to have `null` inside an array of arrays
- This represents an uninitialized element

Null in Arrays

```
double[][] temps = {
    {-10, -20, -15, -16},
    {-20, -23, 0, -3},
    {5, -5, 2, -10},
    null
};

for (int monthIndex = 0; monthIndex < temps.length; monthIndex++){
    double[] monthArr = temps[monthIndex];

    if (monthArr == null){
        System.out.println("null value!");
        continue;
    }

    for (int dayIndex = 0; dayIndex < monthArr.length; dayIndex++){
        System.out.print(monthArr[dayIndex] + ", ");
    }
    System.out.println();
}
//prints
//-10.0, -20.0, -15.0, -16.0,
//-20.0, -23.0, 0.0, -3.0,
//5.0, -5.0, 2.0, -10.0,
//null value!
```

- You can test to see if a reference type is `null`
- It's okay to use == here because we are testing to see if the address is `null`

Inputting Elements in the Array

```
int[][] arr = new int[3][];

System.out.println(Arrays.deepToString(arr));
//prints [null, null, null]

int[] x = {1, 5, 7};
arr[0] = x;

System.out.println(Arrays.deepToString(arr));
//prints [[1, 5, 7], null, null]

arr[0][1] = 99;
System.out.println(Arrays.deepToString(arr));
//prints [[1, 99, 7], null, null]
```

- Can index into the outer and the inner array at once to change elements

Jagged Arrays

`int[][] arr = {{1, 2, 3 }, {0, 0}, {5} }`

The array contains three arrays, of size 3, 2, and 1.

- Sometimes, we may have arrays of arrays of different length
- Be careful on your for-loops that you don't make assumptions about the inner array's length
  - Here we used `monthArr.length` in the inner for-loop

```
for (int monthIndex = 0; monthIndex < temps.length; monthIndex++){
    double[] monthArr = temps[monthIndex];

    for (int dayIndex = 0; dayIndex < monthArr.length; dayIndex++){
        System.out.print(monthArr[dayIndex] + ", ");
    }
    System.out.println();
}
```

Section 5

2D Array Examples

isMatrix

Write a method `isMatrix` that takes a 2D `integer` array as input and returns a `boolean` value.
The method should return true if the 2D array can be read as a matrix, that is, each integer array has the same number of elements. The method returns false otherwise.
Example:

```
int[][] num1 = {{1,2,3}, {5,6}, {8}};
```

`isMatrix(num1)` returns false
and

```
int[][] num2 = {{2,2}, {0,6}, {8,9}};
```

`isMatrix(num2)` returns true

isMatrix Code

```java
public static boolean isMatrix(int[][] a){

    //check for null or empty arrays
    if (a == null || a.length == 0){
        return false;
    }

    //get length of first inner array
    int length = a[0].length;

    //loop through other inner arrays
    for (int i = 1; i < a.length; i++){

        //test if inner array is different size than first array
        if (a[i].length != length){
            return false;
        }
    }

    return true; //all sizes match
}
```

sumMatrix

Write a method sumMatrix that takes two 2D integer arrays as input with the same dimensions. The method should return a new 2D integer array corresponding to their sum.

Example: consider the following 2D arrays

```
int[][] matrix1 = {{2,3}, {5,1}};
int[][] matrix2 = {{-1,5}, {2,-4}};
```

Then sumMatrix(matrix1, matrix2) should return the 2D array

```
{{1, 8}, {7, -3}}
```

sumMatrices

```
public static int[][] sumMatrices(int[][] mx1, int[][] mx2){

    int size = mx1.length;

    int[][] result = new int[size][size];

    for (int i = 0; i < size; i++){
        for (int j = 0; j < size; j++){
            result[i][j] = mx1[i][j] + mx2[i][j];
        }
    }
    return result;
}
```

Section 6

Intro To Objects

Objects

- Objects are the last big concept
- They are a way of organizing our code into different components

For example, a video game might have the following objects:

- Graphics Object
- Sound Object
- Input Object
- Enemy Object

Each of these objects has their own methods

Terminology

- Java *classes* define objects
- We then create *instances* of those classes

For example:

- We have the `Math` class that we can call methods on
- A video-game might have many instances of the `Enemy` class

Objects can refer to:

- Classes, as when we talk about *object-orientated design*
- Or specific instances

Scanner

- Let's look at the **Scanner** class
- This is an **class** that has methods to get input from the user

```
import java.util.Scanner;

public class ScannerExample{

    public static void main(String[] args){

        //create a variable of type Scanner
        Scanner scan = new Scanner(System.in);
    }
}
```

Scanner

- Note that we need `import java.util.Scanner;`
- We create a new *instance* of the `Scanner` class with the new keyword
- We'll talk about the special value `System.in` in a bit
- We create a variable `scan` with the type `Scanner`

```
import java.util.Scanner;

public class ScannerExample{

    public static void main(String[] args){

        //create a variable of type Scanner
        Scanner scan = new Scanner(System.in);
    }
}
```

Scanner

- Let's use the `Scanner` instance to get a number from the user
- We call the `nextInt` method on the `Scanner` instance
- After using the `Scanner`, close it with the `close` method

```
7           //create a variable of type Scanner
8           Scanner scan = new Scanner(System.in);
9
10          //get a number from the user
11          int x = scan.nextInt();
12          System.out.println("You entered: " + x);
13
14          //close the scanner after using it
15          scan.close();
```

Interactions | Console | Compiler Output

```
123
You entered: 123
```

Scanner Methods

These are some of the methods that can be called on a `Scanner` variable to get a value from the user

- `int x = scan.nextInt()` - an integer value
- `double d = scan.nextDouble()` - a double value
- `boolean b = scan.nextBoolean()` - a boolean value

These methods throw an `InputMismatchException` if the user enters the wrong type

Scanner Methods

We also have methods for getting `String` input

- `String s = scan.next()` - a word as a String
- `String line = scan.nextLine()` - a whole line of input

What's the difference?If you enter *hello world* as input:

- `scan.next()` will return *hello* - the next word
- `scan.nextLine()` will return *hello world* - the whole line

Scanner Example

Let's write a program to ask the user for their name and favourite number.

- We'll get the name with the `next()` method
- We'll get the number with the `nextInt()` method

Hello Program

```
public static void sayHello(){

    //create a Scanner variable
    Scanner scan = new Scanner(System.in);

    System.out.println("What is your name and favourite number?");
    String name = scan.next();
    int num = scan.nextInt();

    System.out.println("Hello " + name + "!");
    System.out.println("I also love the number " + num + "!");

    System.out.println("Would you like a nice message?");

    boolean niceMessage = scan.nextBoolean();
    if (niceMessage){
        System.out.println("You are a wonderful person!");
    }

    scan.close();
}
```

Hello Program

```
> run ScannerExample
What is your name and favourite number?
Bentley 8
Hello Bentley!
I also love the number 8!
Would you like a nice message?
true
You are a wonderful person!
```

Guessing Game

Write a program that asks the user to repeatedly guess a random number from 1-100. Tell the user if they guess too high or too low.
When they guess correctly, tell them how many guesses they needed.

- To write this program, we'll use the `Scanner`

Guessing Game

```
//get a random number
int rand = 1 + (int)(Math.random() * 100);
int guess = -1;//choose an initial guess
int tries = 0; //number of guesses

//create the scanner
Scanner scan = new Scanner(System.in);
while (guess != rand){
    System.out.println("Enter a number");
    guess = scan.nextInt();
    tries++; //count the num of tries
    if (guess > rand){
        System.out.println("Too high!");
    }else if (guess < rand){
        System.out.println("Too low!");
    }else{
        System.out.println("You win!");
        System.out.println(tries + " guesses");
    }
}
```

Objects Conclusion

Summary:

- Objects are reference types
- We write Java code to create *classes*
- We create *instances* of *classes* with the new keyword
- We may need to use an *import statement*
- We call methods on instances

Section 7

Random Class

Random Class

- We know how to use `Math.random()` to get random numbers
- But let's see a different way
- Let's use the `Random` class

```
Random randNumGen = new Random();
int rand = randNumGen.nextInt();
System.out.println("Rand num: " + rand);
//printed out Rand num: 1354553317
```

This gives us a random integer from -2.3 billion to 2.3 billion
Note: We need to write `import java.util.Random` to use this class.

Random Exercise

This code uses the `Random` class to generate ten random integers between 1 and 100

`randInt(x)` returns a random number between 0 and x

```
Random randNumGen = new Random();
for (int i=0; i < 10; i++){
    int randNum = 1 + randNumGen.nextInt(100);
    System.out.println("Rand num: " + randNum);
}
```

Sequence of Random Numbers

- What if we wanted to get the same ten random numbers every time we ran the program?
- Why is this useful?
  - Hard to debug code if it does a random thing each time
  - Need to compare output from different students for assignments

To get the same sequence of random numbers, we have to **seed** the generator

Seeding

When creating an instance of `Random`, we can pass in a seed:

```
int seed = 123;//the seed is any number
Random randNumGen = new Random(seed);
for (int i=0; i < 10; i++){
    int randNum = 1 + randNumGen.nextInt(100);
    System.out.println("Seeded rand num: " + randNum);
    //always gives 83, 51, 77, 90, 96, 58, 35...
}
```

Note that the seed isn't the first random number
The seed just picks which sequence of random numbers should be generated

Random Class Methods

`boolean`	**`nextBoolean()`** Returns the next pseudorandom, uniformly distributed `boolean` value from this random number generator's sequence.
`void`	**`nextBytes(byte[] bytes)`** Generates random bytes and places them into a user-supplied byte array.
`double`	**`nextDouble()`** Returns the next pseudorandom, uniformly distributed `double` value between `0.0` and `1.0` from this random number generator's sequence.
`float`	**`nextFloat()`** Returns the next pseudorandom, uniformly distributed `float` value between `0.0` and `1.0` from this random number generator's sequence.
`double`	**`nextGaussian()`** Returns the next pseudorandom, Gaussian ("normally") distributed `double` value with mean `0.0` and standard deviation `1.0` from this random number generator's sequence.
`int`	**`nextInt()`** Returns the next pseudorandom, uniformly distributed `int` value from this random number generator's sequence.
`int`	**`nextInt(int bound)`** Returns a pseudorandom, uniformly distributed `int` value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.