

Lecture March 1 - Intro to Objects

Bentley James Oakes

February 27, 2018

This Lecture

- 1 Midterm
- 2 2D Array Examples
- 3 Intro To Objects
- 4 Random Class
- 5 Tic-Tac-Toe

Section 1

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

- Send me your questions
- And I'll do a short review on Monday, March 12th
- Old exams are on MyCourses

Section 2

2D Array Examples

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

```
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  
}
```


Section 3

Intro To 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

- 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

- 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);
    }
}
```

- 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

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

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

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!
```

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");
    }
}
```

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 4

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 (not including 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.

double

nextDouble()

Returns the next pseudorandom, uniformly distributed `double` 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.

Section 5

Tic-Tac-Toe

Let's start writing a program for playing tic-tac-toe

The program will include an artificial intelligence to play against

Tic-tac-toe

<https://www.google.ca/search?q=tic+tac+toe>

- Create the board
- Display the board
- Write on the board
- Get move from the player
- Get move from the artificial intelligence
- Check if there is a winner
- Play a whole game

Let's start at the top and work our way down

After we can display the board, it's easier to test the other methods

Creating the Board

Let's create a board. We'll have to store the player's symbols on it, so let's create a `char[][]`:

```
public static char[][] createBoard(){
    //create the board
    char[][] board = new char[3][3];

    //initialize it to empty spaces
    for (int row=0; row < board.length; row++){
        for (int col=0; col < board[row].length; col++){
            board[row][col] = ' ';
        }
    }
    return board;
}
```


Drawing the Board

The board might look like this:

```
+ - - - + - - - + - - - +  
|      | X  | O  |  
+ - - - + - - - + - - - +  
|      | X  |    |  
+ - - - + - - - + - - - +  
| O  |    |    |  
+ - - - + - - - + - - - +
```

Drawing the Board Code

```
public static void displayBoard(char[][] board){
    System.out.println("+---+---+---+"); //top line
    for (int row = 0; row < board.length; row++){
        System.out.print("|"); //left edge
        for (int col = 0; col < board[row].length; col++){
            //the spaces and edges
            System.out.print(" " + board[row][col] + " |");
        }
        System.out.println();
        //the line under each row
        System.out.println("+---+---+---+");
    }
}
```

Writing on the Board

```
public static boolean writeOnBoard(char[][] board, int row, int col, char symbol){
    //check to see if the spot is on the board
    if (row < 0 || row >= board.length || col < 0 || col >= board.length){
        return false;
    }

    //check to see if the spot is occupied
    if (board[row][col] != ' '){
        return false;
    }

    //set the spot
    board[row][col] = symbol;

    //return true, the writing was successful
    return true;
}
```

Player's Input

```
public static void getPlayerMove(char[][] board){
    java.util.Scanner scan = new java.util.Scanner(System.in);
    System.out.println("Enter a move: (Example: 1 1)");

    try{
        //get row and col
        int row = scan.nextInt();
        int col = scan.nextInt();

        //write on the board
        writeOnBoard(board, row, col, 'X');
    }catch(java.util.InputMismatchException e){
        //catch any errors in user input
        System.out.println("Invalid input.");
    }
}
```

```
public static void getAIMove(char[][] board){
    //initialize the random num generator
    //don't forget the import statement
    Random rand = new Random();

    //assume we don't have a valid move
    boolean hasValidMove = false;
    while (!hasValidMove){
        //generate a move
        int row = rand.nextInt(3);
        int col = rand.nextInt(3);

        //see if we can write this move to the board
        hasValidMove = writeOnBoard(board, row, col, 'O');
    }
}
```

Check Winner

```
public static char checkWinner(char[][] board){
    //check horizontal
    for (int row = 0; row < board.length; row++){
        if (board[row][0] != ' ' &&
            board[row][0] == board[row][1] &&
            board[row][1] == board[row][2]){
            return board[row][0];
        }
    }
    //check vertical
    for (int col = 0; col < board.length; col++){
        if (board[0][col] != ' ' &&
            board[0][col] == board[1][col] &&
            board[1][col] == board[2][col]){
            return board[0][col];
        }
    }
    //check diagonal
    if (board[0][0] != ' ' &&
        board[0][0] == board[1][1] &&
        board[1][1] == board[2][2]){
        return board[0][0];
    }
    if (board[2][0] != ' ' &&
        board[2][0] == board[1][1] &&
        board[1][1] == board[0][2]){
        return board[2][0];
    }
    return ' '; //no winner
}
```

Play Game

```
public static void playGame(){
    char[][] board = createBoard();

    boolean playing = true;
    while (playing){
        displayBoard(board);
        getPlayerMove(board);
        displayBoard(board);
        getAIMove(board);
        char c = checkWinner(board);
        if (c == 'X'){
            System.out.println("WIN!");
            playing = false;
        }else if (c == 'O'){
            System.out.println("LOSE!");
            playing = false;
        }
    }
}
```

Add more to the tic-tac-toe game:

- Check for ties if there are no empty spots left
- Let the player choose their symbol
- Make a smarter AI
 - Block three-in-a-row
 - Always choose the center on first move