# CS0007 RECITATION

10:00A – 10:50A – LIN ROJTAS

# OVERVIEW

- Method examples: replacing a number
- Recursion
  - Base cases and recursive cases
  - Example: counting down

```java
public static void main(String[] args) {
    int[][] sudoku = {
                        { 0, 4, 0, 0, 0, 0, 1, 7, 9},
                        { 0, 0, 2, 0, 0, 8, 0, 5, 4},
                        { 0, 0, 6, 0, 0, 5, 0, 0, 8},
                        { 0, 8, 0, 0, 7, 0, 9, 1, 0},
                        { 0, 5, 0, 0, 9, 0, 0, 3, 0},
                        { 0, 1, 9, 0, 6, 0, 0, 4, 0},
                        { 3, 0, 0, 4, 0, 0, 7, 0, 0},
                        { 5, 7, 0, 1, 0, 0, 2, 0, 0},
                        { 9, 2, 8, 0, 0, 0, 0, 6, 0}

                    };
    printArray(sudoku);
    System.out.println("Which number would you like to place in the board?");
    Scanner s = new Scanner(System.in);
    int num = s.nextInt();
    System.out.println("Which row would you like to place this number in?");
    int rowInd = s.nextInt();
    System.out.println("Which column would you like to place this number in?");
    int colInd = s.nextInt();

    int[][] newSudoku = replaceNum(sudoku, num, rowInd, colInd);
    printArray(newSudoku);

}
```

```java
public static int[][] replaceNum(int[][] arr, int numToReplace, int row, int col) {
    int[][] newArr = arr;
    newArr[row][col] = numToReplace;
    return newArr;
}

public static void printArray(int[][] arr) {
    System.out.println("The sudoku board is:");
    for (int i = 0; i < arr.length; i++) {
        for (int j = 0; j < arr[0].length; j++) {
            System.out.print(arr[i][j] + " ");
        }
        System.out.println();
    }
    System.out.println("=========================================================");
}
```

- Three methods
  - main
  - replaceNum (non void)
  - printArray (void)
- Let's see how the two non-main methods work!

- The first method that we call is printArray
- The printArray method is a *void* method that takes in a two-dimensional array as a parameter
- The code in the main method will stop until the printArray method is done running

```java
public static void main(String[] args) {
    int[][] sudoku = {
                        { 0, 4, 0, 0, 0, 0, 1, 7, 9},
                        { 0, 0, 2, 0, 0, 8, 0, 5, 4},
                        { 0, 0, 6, 0, 0, 5, 0, 0, 8},
                        { 0, 8, 0, 0, 7, 0, 9, 1, 0},
                        { 0, 5, 0, 0, 9, 0, 0, 3, 0},
                        { 0, 1, 9, 0, 6, 0, 0, 4, 0},
                        { 3, 0, 0, 4, 0, 0, 7, 0, 0},
                        { 5, 7, 0, 1, 0, 0, 2, 0, 0},
                        { 9, 2, 8, 0, 0, 0, 0, 6, 0}

                    };
    printArray(sudoku);
    System.out.println("Which number would you like to place in the board?");
    Scanner s = new Scanner(System.in);
    int num = s.nextInt();
    System.out.println("Which row would you like to place this number in?");
    int rowInd = s.nextInt();
    System.out.println("Which column would you like to place this number in?");
    int colInd = s.nextInt();

    int[][] newSudoku = replaceNum(sudoku, num, rowInd, colInd);
    printArray(newSudoku);

}
```

```java
public static void printArray(int[][] arr) {
    System.out.println("The sudoku board is:");
    for (int i = 0; i < arr.length; i++) {
        for (int j = 0; j < arr[0].length; j++) {
            System.out.print(arr[i][j] + " ");
        }
        System.out.println();
    }
    System.out.println("=====================================================");
}
```

```
The sudoku board is:
0 4 0 0 0 0 1 7 9
0 0 2 0 0 8 0 5 4
0 0 6 0 0 5 0 0 8
0 8 0 0 7 0 9 1 0
0 5 0 0 9 0 0 3 0
0 1 9 0 6 0 0 4 0
3 0 0 4 0 0 7 0 0
5 7 0 1 0 0 2 0 0
9 2 8 0 0 0 0 6 0
=====================================================
```

- Within this method, we print out “The sudoku board is:” followed by each number in the array in the same way we display any other two-dimensional array.
- This is a void method, so there is no return keyword. Also, we do not need to assign calls to this method to a new variable in main.

- Now that printArray is done running, we pick up where we left off with these print statements and taking in user inputs.
- A general note: Scanners can be reused! You don't need to create a new one for each new user input

```java
public static void main(String[] args) {
    int[][] sudoku = {
                        { 0, 4, 0, 0, 0, 0, 1, 7, 9},
                        { 0, 0, 2, 0, 0, 8, 0, 5, 4},
                        { 0, 0, 6, 0, 0, 5, 0, 0, 8},
                        { 0, 8, 0, 0, 7, 0, 9, 1, 0},
                        { 0, 5, 0, 0, 9, 0, 0, 3, 0},
                        { 0, 1, 9, 0, 6, 0, 0, 4, 0},
                        { 3, 0, 0, 4, 0, 0, 7, 0, 0},
                        { 5, 7, 0, 1, 0, 0, 2, 0, 0},
                        { 9, 2, 8, 0, 0, 0, 0, 6, 0}

                    };
    printArray(sudoku);
    System.out.println("Which number would you like to place in the board?");
    Scanner s = new Scanner(System.in);
    int num = s.nextInt();
    System.out.println("Which row would you like to place this number in?");
    int rowInd = s.nextInt();
    System.out.println("Which column would you like to place this number in?");
    int colInd = s.nextInt();

    int[][] newSudoku = replaceNum(sudoku, num, rowInd, colInd);
    printArray(newSudoku);

}
```

```java
System.out.println("Which number would you like to place in the board?");
Scanner s = new Scanner(System.in);
int num = s.nextInt();
System.out.println("Which row would you like to place this number in?");
int rowInd = s.nextInt();
System.out.println("Which column would you like to place this number in?");
int colInd = s.nextInt();
```

```
Which number would you like to place in the board?
8
Which row would you like to place this number in?
0
Which column would you like to place this number in?
0
```

- Nothing new is going on here, we are just declaring variables based on user input.
- The integer num now stores 8
- The integer rowInd now stores 0
- The integer colInd now stores 0

- The next method that we call is replaceNum
- The replaceNum method is a *non-void* method that takes in a two-dimensional array and three integers as parameters
- The code in the main method will stop until the replaceNum method is done running

```java
public static void main(String[] args) {
    int[][] sudoku = {
                        { 0, 4, 0, 0, 0, 0, 1, 7, 9},
                        { 0, 0, 2, 0, 0, 8, 0, 5, 4},
                        { 0, 0, 6, 0, 0, 5, 0, 0, 8},
                        { 0, 8, 0, 0, 7, 0, 9, 1, 0},
                        { 0, 5, 0, 0, 9, 0, 0, 3, 0},
                        { 0, 1, 9, 0, 6, 0, 0, 4, 0},
                        { 3, 0, 0, 4, 0, 0, 7, 0, 0},
                        { 5, 7, 0, 1, 0, 0, 2, 0, 0},
                        { 9, 2, 8, 0, 0, 0, 0, 6, 0}

                    };
    printArray(sudoku);
    System.out.println("Which number would you like to place in the board?");
    Scanner s = new Scanner(System.in);
    int num = s.nextInt();
    System.out.println("Which row would you like to place this number in?");
    int rowInd = s.nextInt();
    System.out.println("Which column would you like to place this number in?");
    int colInd = s.nextInt();

    int[][] newSudoku = replaceNum(sudoku, num, rowInd, colInd);
    printArray(newSudoku);

}
```

- Note that you do not need to name the variables in your method call the same names as your parameters!
  - You *can* do this, but you do not have to. Just make sure you use the parameter variable names within the actual method's code
- We call the process of inputting specific values or variables into a method *passing*
  - i.e. here, we are passing the values of sudoku, num, rowInd, and colInd into the method replaceNum

```
{ 0, 4, 0, 0, 0, 0, 1, 7, 9},
{ 0, 0, 2, 0, 0, 8, 0, 5, 4},
{ 0, 0, 6, 0, 0, 5, 0, 0, 8},
{ 0, 8, 0, 0, 7, 0, 9, 1, 0},
{ 0, 5, 0, 0, 9, 0, 0, 3, 0},
{ 0, 1, 9, 0, 6, 0, 0, 4, 0},
{ 3, 0, 0, 4, 0, 0, 7, 0, 0},
{ 5, 7, 0, 1, 0, 0, 2, 0, 0},
{ 9, 2, 8, 0, 0, 0, 0, 6, 0}
```

8 0 0

```
replaceNum(sudoku, num, rowInd, colInd);
```

```
public static int[][] replaceNum(int[][] arr, int numToReplace, int row, int col) {
    int[][] newArr = arr;
    newArr[row][col] = numToReplace;
    return newArr;
}
```

- First, we create a new array called newArr that is equal to the array that we passed in.
- Then, we use the inputted row and column value to replace whatever number was in that position before with the user-inputted number
- Finally, we **return** the new array since this method is a non-void method; the return type is a two-dimensional array.
  - The return type and the type of variable in the return statement MUST be the same.

```
replaceNum(sudoku, num, rowInd, colInd);
```

```
public static int[][] replaceNum(int[][] arr, int numToReplace, int row, int col) {
    int[][] newArr = arr;
    newArr[row][col] = numToReplace;
    return newArr;
}
```

- This is the new value of newArr, which is what we are returning to the main method and storing in variable newSudoku

```
{ 8, 4, 0, 0, 0, 0, 1, 7, 9},
{ 0, 0, 2, 0, 0, 8, 0, 5, 4},
{ 0, 0, 6, 0, 0, 5, 0, 0, 8},
{ 0, 8, 0, 0, 7, 0, 9, 1, 0},
{ 0, 5, 0, 0, 9, 0, 0, 3, 0},
{ 0, 1, 9, 0, 6, 0, 0, 4, 0},
{ 3, 0, 0, 4, 0, 0, 7, 0, 0},
{ 5, 7, 0, 1, 0, 0, 2, 0, 0},
{ 9, 2, 8, 0, 0, 0, 0, 6, 0}
```

- Note again that since replaceNum is a non-void method, we need to store its return value in a variable in the main method
- Finally, we call printArray on newSudoku so the new sudoku board with the updated number can be displayed.

```java
public static void main(String[] args) {
    int[][] sudoku = {
                        { 0, 4, 0, 0, 0, 0, 1, 7, 9},
                        { 0, 0, 2, 0, 0, 8, 0, 5, 4},
                        { 0, 0, 6, 0, 0, 5, 0, 0, 8},
                        { 0, 8, 0, 0, 7, 0, 9, 1, 0},
                        { 0, 5, 0, 0, 9, 0, 0, 3, 0},
                        { 0, 1, 9, 0, 6, 0, 0, 4, 0},
                        { 3, 0, 0, 4, 0, 0, 7, 0, 0},
                        { 5, 7, 0, 1, 0, 0, 2, 0, 0},
                        { 9, 2, 8, 0, 0, 0, 0, 6, 0}

                    };
    printArray(sudoku);
    System.out.println("Which number would you like to place in the board?");
    Scanner s = new Scanner(System.in);
    int num = s.nextInt();
    System.out.println("Which row would you like to place this number in?");
    int rowInd = s.nextInt();
    System.out.println("Which column would you like to place this number in?");
    int colInd = s.nextInt();

    int[][] newSudoku = replaceNum(sudoku, num, rowInd, colInd);
    printArray(newSudoku);

}
```

```java
public static void printArray(int[][] arr) {
    System.out.println("The sudoku board is:");
    for (int i = 0; i < arr.length; i++) {
        for (int j = 0; j < arr[0].length; j++) {
            System.out.print(arr[i][j] + " ");
        }
        System.out.println();
    }
    System.out.println("=====================================================");
}
```

```
The sudoku board is:
8 4 0 0 0 0 1 7 9
0 0 2 0 0 8 0 5 4
0 0 6 0 0 5 0 0 8
0 8 0 0 7 0 9 1 0
0 5 0 0 9 0 0 3 0
0 1 9 0 6 0 0 4 0
3 0 0 4 0 0 7 0 0
5 7 0 1 0 0 2 0 0
9 2 8 0 0 0 0 6 0
=====================================================
```

- Within this method, we print out “The sudoku board is:” followed by each number in the array in the same way we display any other two-dimensional array.
- This is a void method, so there is no return keyword. Also, we do not need to assign calls to this method to a new variable in main.
- With this, we have reached the end of our program.

# RECURSION: AN OVERVIEW

- Recursion is the process of making a method call itself
- This provides a way to break larger problems down into simple subproblems.
- When writing a recursive method, you must ensure that you have:
  - A base case, otherwise known as a halting case, that is attainable
  - A recursive case in which the method calls itself

# RECURSION: AN EXAMPLE

- Recursion can be difficult to wrap your head around, so we're going to explore this recursive method that adds a range of numbers together
  - (i.e. 5+4+3+2+1)

```java
public static void main(String[] args) {
    int result = sum(5);
    System.out.println(result);
}

public static int sum(int k) {
    if (k > 0) {
        return k + sum(k - 1);
    } else {
        return 0;
    }
}
```

Recursive case

Base case

- First, the sum method is called; sum is a non-void method that returns an integer as well as taking an integer in as a parameter.
- We call sum(5), which takes us down to the sum method
  - In this situation, k = 5
  - If k > 0, we return k + sum(k-1)
  - In other words, we return 5 + sum(4)…

```java
public static void main(String[] args) {
    int result = sum(5);
    System.out.println(result);
}

public static int sum(int k) {
    if (k > 0) {
        return k + sum(k - 1);
    } else {
        return 0;
    }
}
```

- Since sum(4) was called in our last return statement, we go through sum again with k = 4.
- We call sum(4), which takes us back to the sum method
  - In this situation, k = 4
  - If k > 0, we return k + sum(k-1)
  - In other words, we return 4 + sum(3)…

```java
public static void main(String[] args) {
    int result = sum(5);
    System.out.println(result);
}

public static int sum(int k) {
    if (k > 0) {
        return k + sum(k - 1);
    } else {
        return 0;
    }
}
```

- Since sum(3) was called in our last return statement, we go through sum again with k = 3.
- We call sum(3)… this is getting repetitive
  - sum(3) will return 3 + sum(2)
  - sum(2) will return 2 + sum(1)
  - sum(1) will return 1 + sum(0)
- When sum(0) is called, k = 0… which isn't greater than 0!
  - We've hit our base case! Return 0!

```java
public static void main(String[] args) {
    int result = sum(5);
    System.out.println(result);
}

public static int sum(int k) {
    if (k > 0) {
        return k + sum(k - 1);
    } else {
        return 0;
    }
}
```

- We're not done yet!!!!!
- There is still the matter of all those other method calls to go…
  - sum(0) = 0
  - sum(1) = 1 + sum(0) = 1 + 0 = 1
  - sum(2) = 2 + sum(1) = 2 + 1 = 3
  - sum(3) = 3 + sum(2) = 3 + 3 = 6
  - sum(4) = 4 + sum(3) = 4 + 6 = 10
  - sum(5) = 5 + sum(4) = 5 + 10 = 15!
- Thus, result = 15, so 15 will be printed.

```java
public static void main(String[] args) {
    int result = sum(5);
    System.out.println(result);
}

public static int sum(int k) {
    if (k > 0) {
        return k + sum(k - 1);
    } else {
        return 0;
    }
}
```

```
PS C:\Users\lrojt\Documents> java Test
15
```

# WHY RECURSION?

- Despite it taking up more memory than iterative methods (i.e., ones that contain loops), recursion can reduce the amount of time that it takes to do certain problems
  - The most important of these is sorting; if you choose to move forward in your computer science career, you'll learn about sorting algorithms in future classes. The fastest ones use recursion!
- At its core, recursion is essentially dividing one large problem into several smaller subproblems until they are manageable; I'm sure you've done something similar on homework assignments!

# FOR NEXT WEEK

- Next week will be extended office hours in preparation for your final exam, but I'll be bringing candy as a reward for your good work this semester
- All the labs that you may be missing are due Monday, December 6th at 11:59PM.
- Don't forget to mask up and wash your hands! These new variants don't mess around