Lecture 17

For Array Class Shenanigans

For or While?

```
class WhileDemo {
    public static void main(String[] args){
        int count = 1;
        while (count < 11) {
            System.out.println("Count is: " + count);
            count++;
        }
    }
}
```

Note: They both evaluate the predicate expression (boolean check) before proceeding to the enclosed code block.

```
class ForLoopDemo {
    public static void main(String[] args {
         for(int count = 1; count < 11; count++){
              System.out.println("Count is: " + count);
         }
    }
}
```

For or While?

```
class WhileDemo {
    public static void main(String[] args){
        int count = 1;
        while (count != 0) {
            System.out.println("Count is: " + count);
            count = keyboard.nextInt();
        }
    }
}
```

Note: These are both valid pieces of code.

```
class ForLoopDemo {
    public static void main(String[] args {
         for(int count = 1; count != 0; ){
              System.out.println("Count is: " + count);
              count = keyboard.nextInt();
         }
    }
}
```

Array Reminder

So far, you have been working with **variables** that hold only **one value**. The **integer** variables you have set up have held only **one number** (and *later* in the quarter we will see how **string** variables will hold **one long string** of text).

An **array** is a collection to hold *more than one value at a time*. It's like a **list** of items.

Think of an array as like the columns in a spreadsheet. You can have a spreadsheet with only one column, or several columns.

The data held in a single-list (one-dimensional) array might look like this:

	grades
0	100
1	89
2	96
3	100
4	98

```
grades  = new int[5];
```

grades is a named reserved space set aside to hold exactly five **[5]** 32-bit elements all initializing to a value of zero **0**. As we have learned about programming languages, the “index” always starts at **0**, not **1**, and procedes until the size of the array is reached. In our example, since we declared **[5]** the array element index starts with **0** and ends at **4.**

array element index →	0	1	2	3	4
space reserved for data →	32-bits	32-bits	32-bit	32-bits	32-bits
value initialized in element →	0	0	0	0	0

index	grades
0	100
1	89
2	96
3	100
4	98

```
grades  = new int[5]; // <-- Steps 1 & 2

grades[0] = 100; // Steps 3
grades[1] = 89;
grades[2] = 96;
grades[3] = 100;
grades[4] = 98;
```

STEP 1: Declare Variable
STEP 2: Allocate Memory
STEP 3: Initialize Elements

5

grades

0	1	2	3	4
100	89	96	100	98

About Array Sizes

- The array size must be a **non-negative number**.
- It may be a **literal value**, a **constant**, or **variable**.

```
final int ARRAY_SIZE = 6;
int[] numbers = new int[ARRAY_SIZE];
```

- Once created, an array size is **fixed** and **cannot** be changed.

Off-by-one Errors

- It is very easy to be “**off-by-one**” when accessing arrays

```
// This code has an off-by-one error.
int[] numbers = new int[100];
for (int i = 1; i <= 100; i++)
     // Would work with < only
{
   numbers[i] = 99;
}
```

- Here, the equal sign allows the loop to continue on to index **100**, but **99** is the last index in the array

- This code would throw an **ArrayIndexOutOfBoundsException**

More Array Declarations

Previously we showed arrays being declared:

```
    int[] numbers;
```

However, the brackets can also go here:

```
    int numbers[];
```

These are *equivalent* but the first style is **typical** (and preferred by most developers/coders).

Multiple arrays can be declared on the same line.

```
    int[] numbers, codes, scores;
```

With the *alternate* notation each variable must have brackets.

```
int numbers[], codes[], scores;
```

The **scores** variable in this instance is simply an **int** variable.

Array of Objects Declaration

```
Card[] deck = new Card [52];  // declares an array of type Card
```

- In this case, each element is initialized to null pointer.
- SO, we need to initialize each element and give it a value

```
int index = 0;
for (int suit = 0; suit < 4; suit++) {
  for (int rank = 1; rank <= 13; rank++) {
    deck[index] = new Card (suit, rank); // call the constructor
    index++;
  }
}
```

Enhanced For-loop

- Simplified array processing (read only)
- Always goes through all elements
- General:

```
for(datatype elementVariable : array) {
    statement;
}
```

```
int[] numbers = {3, 6, 9};
for(int val : numbers) { // <-- Only two parts. You can read the line as
    // "iterate on elements from the collection named numbers. The current
    //  element will be referenced by the int val."
    System.out.println("The next value is " + val);
}
```

Encapsulation

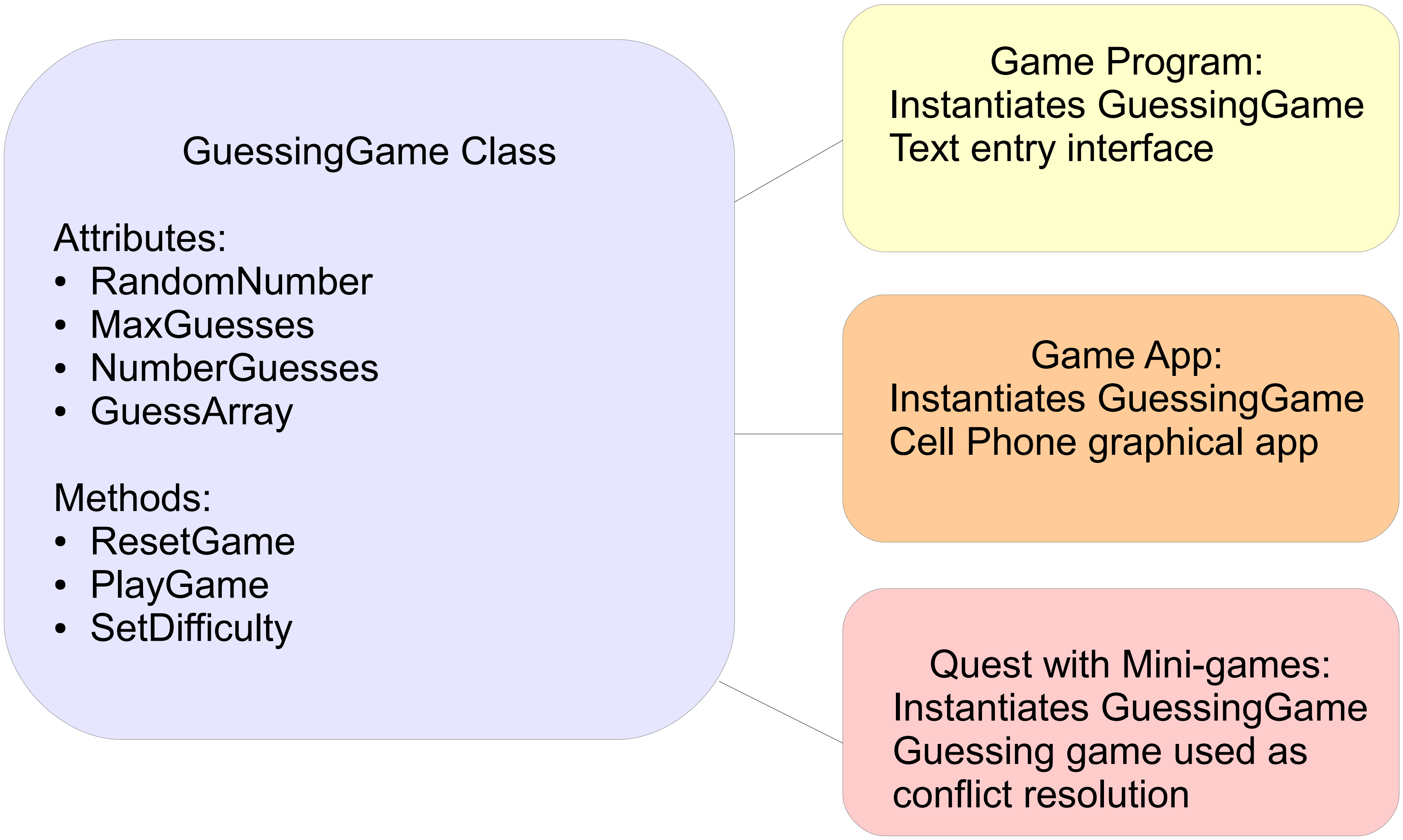
Encapsulate: to show or express the main idea or quality of (something) in a brief way,

to completely cover (something) especially so that it will not touch anything else

In programming, encapsulation refers to the bundling of data with the methods that operate on that data.

- Groups related data and methods
- Suggests protecting (making private) object attributes
- Reduces collisions of like-named variables
- Allows for refactoring by making code segments independent

Class Separation



GuessingGame Class

Attributes:

- RandomNumber
- MaxGuesses
- NumberGuesses
- GuessArray

Methods:

- ResetGame
- PlayGame
- SetDifficulty

Game Program:
Instantiates GuessingGame
Text entry interface

Game App:
Instantiates GuessingGame
Cell Phone graphical app

Quest with Mini-games:
Instantiates GuessingGame
Guessing game used as conflict resolution