CMPINF0401 Recitation

TUESDAYS 11:00-12:50

MICHAEL BARTLETT

Overview

- Methods
- Array loading with loops and files
- Lab 6
- Assignment 3
- Midterm Review

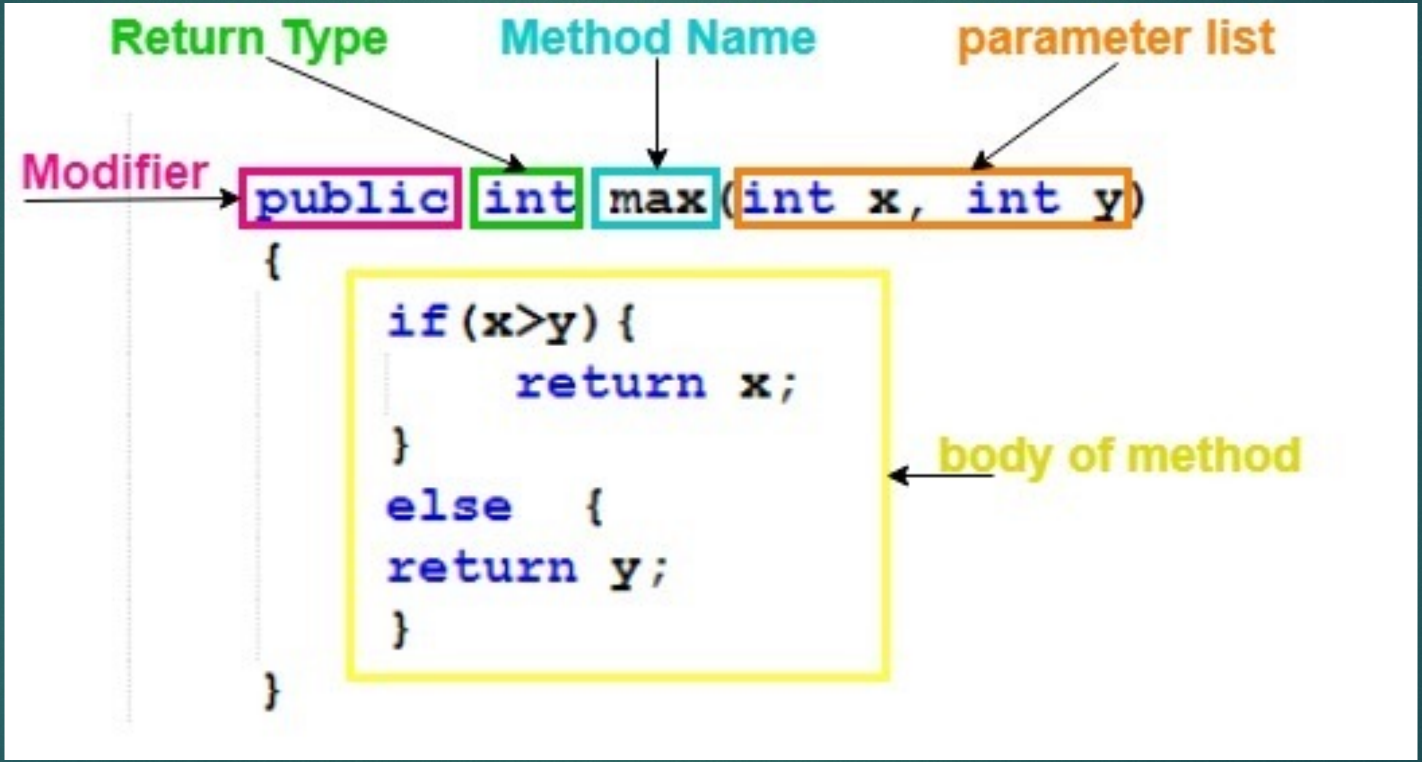
What is a Method?

- A method is a block of code that only runs when it is called
- Up until now, we’ve been writing all of our code in one contiguous block
  - As we write larger programs, this won’t hold up
  - It gets hard to figure out what code does where, so that’s where methods come in!
- You can also think of them as a program within a program

Where Have We Seen These Before?

- Lots of places!
  - `public static void main (String args[])` is a method declaration!
- Remember how we (repeatedly) talked about classes/objects (like String or Scanner for example) have their own special set of instructions?
  - Those instructions are methods!
  - Scanner.nextLine(), String.toUpperCase(), String.substring(1, 2), and many many more

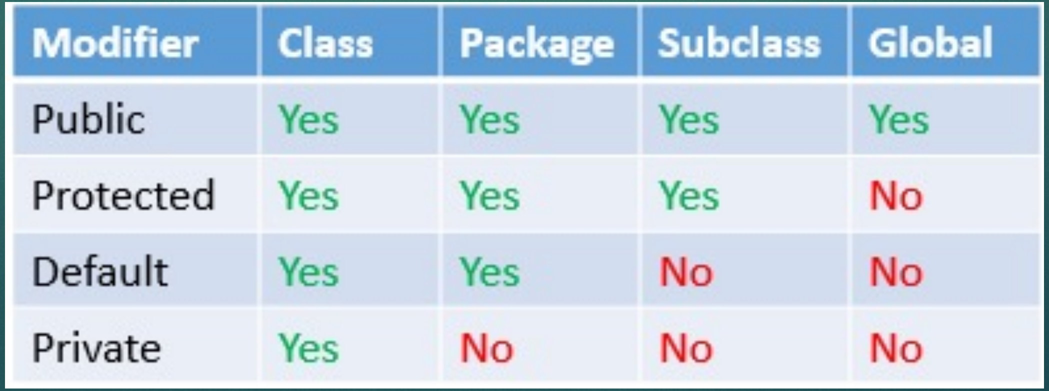
Methods: A Breakdown



```
public int max(int x, int y)
{
    if(x>y){
        return x;
    }
    else  {
    return y;
    }
}
```

Methods: Modifiers and Accessibility

- The first part of any method that we write is the accessibility modifier
  - This will be either public, private, protected, or… nothing at all!
  - Different modifiers mean different things for which programs can access your methods
  - In terms of accessibility, `public > protected > default > private`
  - Most of the time, we'll be working with public and private, but the other two are good to know



Modifier	Class	Package	Subclass	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

Methods: Modifiers and Accessibility

- Following your access modifier will usually be your return type (more on that in a moment), but sometimes you'll have another modifier unrelated to privacy
  - Final, static, or nothing at all!
- The final keyword means that a method cannot be overwritten
  - Similar to how a final variable cannot be changed…

Methods: Modifiers and Accessibility

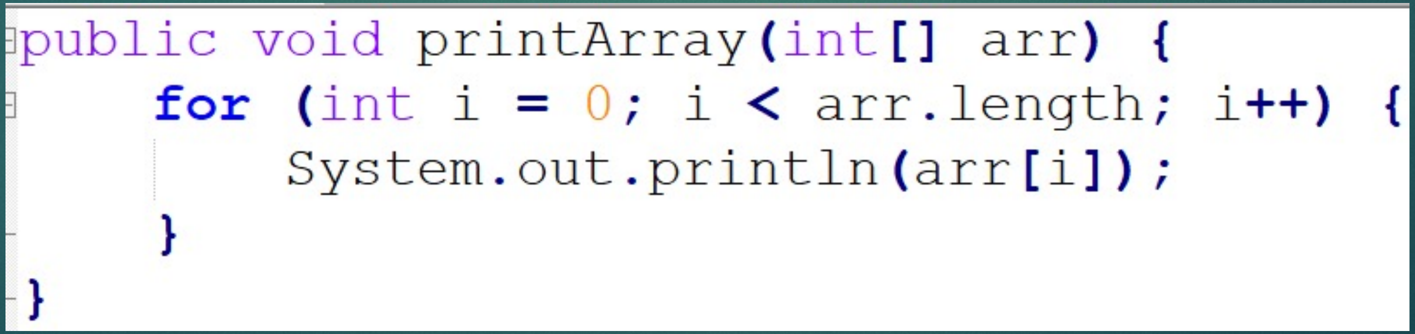
- A static method is a method that does not require an instance of a certain class
  - In other words… when you're calling a static method, you'll use the name of the class rather than creating a variable/object for it.
  - Example: `Math.pow(2, 3);`
- Non-static methods require an instance of the class in which they belong to in order to have functionality
  - In other words… we create an object and use the name of the variable when we're calling it
  - Example: `String str = "computer"; str.toUpperCase();`

Return Types

- Following any modifiers (access or otherwise) will be the method's return type
  - This will be the type of thing (primitive or object) that the method returns.
  - `public int methodName()` returns an int!
  - `public static String firstName()` returns a String!
  - `private final double length()` returns a double!
- A return type can be seen as "the final product" after utilizing a method

Return Types

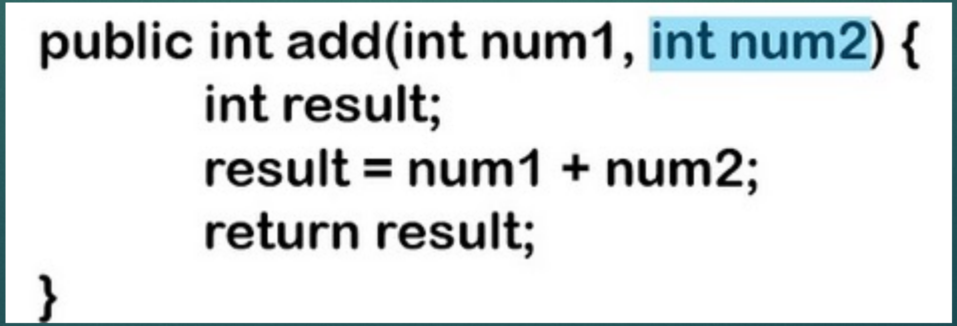
- Sometimes, a method will do something, but it won’t return anything
  - These are called void methods
- Void methods might be used for a variety of purposes, but one of the most common is outputting lots of print statements



```
public void printArray(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        System.out.println(arr[i]);
    }
}
```

Return Types

- We call this part of the method a return type in part because of the return keyword in Java
  - Some time in the method (sometimes it will be at the end, sometimes it won't) we will use the return keyword to tell the method what exactly it will give the programmer/user
  - We may have more than one return statement, but **void methods have no return statements**
  - When called, the method below will give back the sum of the two numbers, which is stored in the `result` variable. Thus, we `return result;`



```
public int add(int num1, int num2) {
        int result;
        result = num1 + num2;
        return result;
}
```

Method Naming Conventions

- Method naming conventions will be very similar to how you name variables
  - First word is all lowercase
  - If there is more than one word to your method name, all the following words will begin with capital letters
  - Use alphanumeric characters only, don't begin with a number
  - Include parentheses at the end – more on what they mean in a bit!
- Examples:
  - `public void printArray()`
  - `private int mySuperCoolMethod()`
  - `public double max()`

Parameters

- Parameters (or arguments) are essentially the **input value** of a method
  - Not every method will have parameters, but some will!
  - When you're declaring your method, you'll need to declare the parameters as well if you have them
- For example, with the substring method:
  - `str.substring(1, 3);` ← those numbers are parameters of the method substring!
- The method declaration for substring may look something like this:
  - `public String substring(int start, int end);`

Parameters and Local/Global Variables

- The parameters that you declare in your method declaration will only be able to be used *within* your method
  - As such, these are called local variables
  - We've used local variables before, specifically when working with for loops!
    - `for (int i = 0; i < 10; i++)` ← i is a local variable!
- Any variable declared outside any methods is considered a global variable

Loading Arrays with Files

- Declare an array of some size to be used to load the data into it
  - Typically it's made to be bigger than your data set so that you don't have to worry about upsizing, just trimming after the fact.
- Read the file and while(file.hasNext()), keep reading the file into the array and use your count variable as reference of where you are.
  - If your array wasn't big enough you'd have to upsize it.
    - This is done by making a new array (typically double the size) and taking all the elements that were in the original array and putting them in the array of bigger size before continuing to read the values in.
      - Can be seen in Lab6.
- Once the array is filled, trim as needed.
- FiletoArray.java
  - Credit: Prof. Devine (on Canvas)

Lab 6

- Lab6.java
- Base your solution off the trim method from FiletoArray.java

Assignment 3

- Item 1: Simple if statement to check and make sure args.length < 1.
  - If args.length < 1, allow the user to know the proper usage of opening the file and then exit the program.
    - This is seen in FiletoArray.java
- Item 2: Make a new array of size MAX_CAPACITY and set a count variable = 0.
- Item 3: Make a new Scanner object using the filename that was passed in (args[0]) and then use a while loop to loop through the file while there is a next int.
  - Item 4: Inside this while loop call your insertInOrder method (while passing the proper args) and then increment your count.
    - Void method so no need to set anything equal to the method call

Assignment 3

- Item 4: Insert in Order:
  - Declare an uninitialized counting variable (i for example) to be used for a for loop
  - In the for loop, set i = count – 1 (count will not work because that's the total number of elements, we need to go to the value last stored)
  - Condition for the for loop: Make sure that i >= 0 and that newVal is less than the current val we are at
  - Decrement i since we're starting at the high end of the array
  - Once this for loop breaks, you can set your arr[i + 1] = newVal
- Item 5: After the loop, call your trimArray method (use your lab6 as a base for writing this method) and set your array equal to the array that this method is returning
- Item 6: Call your print array method.
  - Remember, we can't just call System.out.println(arrayName), instead you need to loop through the array and print out each value. You can see this in last week's slides.
- A3.java

Midterm Review

- Midterm topics