Today we’ll cover...

- Review nested loops
- Method Signatures & Parameter Passing
- Method Overloading
- Lab4
Nested Loops

What does this code do?

```java
for(int i = 0; i < 5; i++) {
    for(int j = 0; j < 5; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```
Output

*****

*****

*****

*****

*****

*****
Nested Loops

What does this code do?

```java
for(int i = 0; i < 5; i++) {
    for(int j = i; j < 5; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```
Output

******

****

***

**

*
Nested Loops

What does this code do?

```java
for(int i = 0; i < 5; i++) {
    for(int j = 0; j <= i; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```
Output

*

**

***

****

*****

******
Printing Multiplication Tables

- Take a moment to think about how you might write a program to print out the multiplication tables up to 5…

- Solution:

```java
for(int i = 1; i <= 5; i++) {
    for(int j = 1; j <= 5; j++) {
        System.out.print( (i*j) + " ");
    }
    System.out.println();
}
```
public static void printMultTable(int max) {
    for(int i = 0; i <= max; i++) {
        if(i == 0)
            System.out.print("X ");
        else if(i < 10)
            System.out.print(i + " ");
        else
            System.out.print(i);

        for(int j = 1; j <= max; j++) {
            if(i == 0)    //print column header
                System.out.printf("%4d", j);
            else
                System.out.printf("%4d", (i * j));
        }
        System.out.println();
    }
    System.out.println();
}
Writing a Method

public static void main(String[] args) {
    //some code here
}

<modifier(s)> <return-type> <method-name>(<formal-params>) {
    <method-body>
    return <return-value>
}

Writing a Method - Modifiers

- **Visibility Modifiers**
  - public, private, protected, default
  - A visibility modifier specifies how data fields and methods in a class can be accessed from outside the class.
  - More on this when we talk about objects/classes

- **Non-Access Modifiers**
  - static, final, abstract, volatile, synchronized
    - final makes a variable an unchangeable constant
    - volatile and synchronized have to do with threads (we won’t use them in this class)
    - More on static and abstract in upcoming lectures
Writing a Method - Return Type

- Specify what type of value this method will return
  - Primitives
  - Objects
  - void (no return type)
Writing a Method - Method Name

- Use descriptive names that describe what the method does
- Use camelCase just like when naming variables

```java
public static void printHello() {
    System.out.println("Hello!");
}

public static int max(int a, int b) {
    return (a > b) ? a : b;
}
```
Writing a Method - Parameters

- The parameters in the method signature are called “formal parameters” whereas the values passed as parameters when the method is called are “actual parameters.”

```java
public static int max(int a, int b) {
    // a and b are the formal params
    return (a > b) ? a : b;
}

max(4, 5); // 4 and 5 are the actual params
```
Method Overloading

- A method is “overloaded” when a method shares the same name with a different signature (i.e., return type/params)
  - A method must have different parameters to be overloaded; changing the modifiers/return-type alone do not qualify

```java
public static int sum(int a, int b) {
    return a + b;
}

public static double sum(double a, double b) {
    return a + b;
}
```
Ambiguous Method Overloading

- An overloaded method may appear ambiguous if the compiler doesn’t know which method to call

```java
public static double sum(int a, double b) {
    return a + b;
}

public static double sum(double a, int b) {
    return a + b;
}

sum(1, 2); //compiler will throw an error b/c invocation is ambiguous
```
Pass By Value

- When you invoke a method with an argument, the value of the argument is passed to the parameter. This is referred to as pass-by-value.
- If the argument is a variable rather than a literal value, the value of the variable is passed to the parameter. The variable is not affected, regardless of the changes made to the parameter inside the method.

- Java is Pass-By-Value for ALL types (primitives and reference types (i.e., objects))
  - However, the “value” of an object is a reference to that object on the heap. Thus, object data members can be modified inside of a method if said data members are accessible
Lab4 - Car

- Write a program that simulates a car moving around a 20x20 grid
- The car has three attributes:
  - Color, Ignition, Position (represented by an X coordinate and Y coordinate)
  - The color and initial position should be randomly assigned at the start of the program
  - Ignition should be set to false (i.e., off) initially
- The user is then prompted to move the car around grid, but the car can only move within the bound of the grid and only when the car is on
- The program should run indefinitely until the user decides to quit
- Specific error handling and input validation will be expected (more details in the lab specification)