Kaitlin Hoffmann

Office Hours:

SH 243 MR 11:00 - 12:30 PM via appointment https://calendly.com/hoffmank4/15min

Email: hoffmank4@newpaltz.edu

For TA Office Hours and Email – Please see syllabus

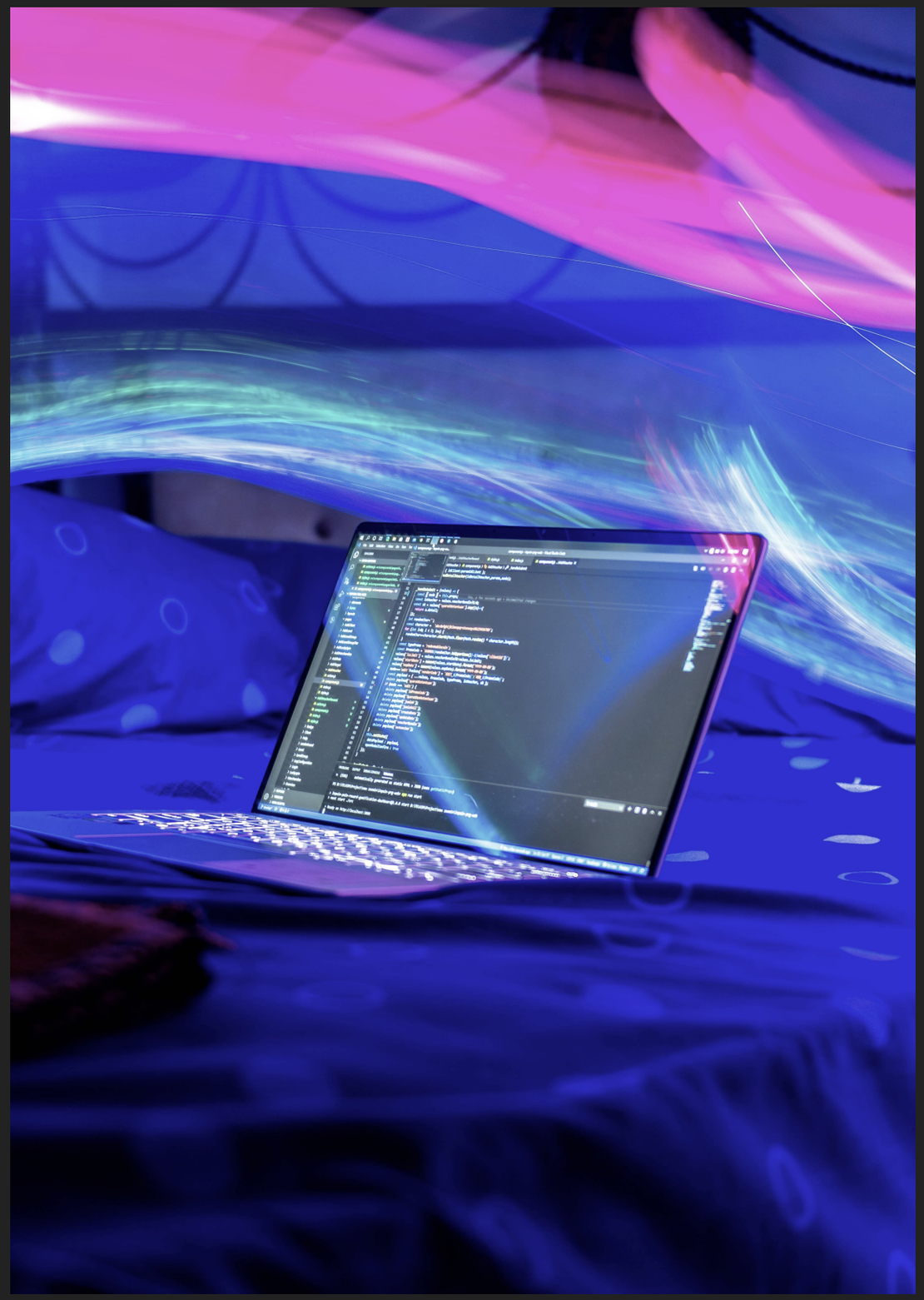
NEW! Supplemental Instruction: Sign up at my.newpaltz.edu

METHODS PART 2

COMPUTER SCIENCE I

OBJECTIVES

- More practice
- Modularizing code
- The Scanner class
- The Math class



REVIEW – WRITE A METHOD HEADER FOR THE FOLLOWING:

- Compute 1+2+3+...+n
- Write a program that asks a user for a three-digit integer, and determines whether it is a palindrome number.
- Write a program that reads three edges for a triangle and computes the perimeter if the input is valid. Otherwise, display that the input is invalid. The input is valid if the sum of every pair of two edges is greater than the remaining edge.

REVIEW – WRITE A METHOD HEADER FOR THE FOLLOWING:

- Compute 1+2+3+...+n

```
public static int findSum(int num) {
    int sum = 0;
    for(int i = 0; i<=num; i++) {
        sum+=i;
    }
    return sum;
}
```

REVIEW – WRITE A METHOD HEADER FOR THE FOLLOWING:

- Write a program that asks a user for a three-digit integer, and determines whether it is a palindrome number.

```
public static boolean palindrome(int num) {
    int ones = num%10;
    int hundreds = num/100;

    return ones == hundreds;
}
```

REVIEW – WRITE A METHOD HEADER FOR THE FOLLOWING:

- Write a program that reads three edges for a triangle and computes the perimeter if the input is valid. Otherwise, display that the input is invalid. The input is valid if the sum of every pair of two edges is greater than the remaining edge.

```
public static double getPerimeter(double d1, double d2, double d3) {
    if(d1 < d2 + d3 && d2 < d1 + d3 && d3 < d1 + d2) {
        return d1 + d2 + d3;
    } else {
        System.out.println("Input invalid");
        return -1;
    }
}
```

CAN WE USE METHODS INSIDE OTHER METHODS?

- **Of course!** If you think about it, the main header is a method, and we use methods inside it.
- It's a good idea to break up complex problems into multiple methods for a variety of reasons which we will get into in a few slides.
- But first, let's look at an example…

CAN WE USE METHODS INSIDE OTHER METHODS?

```
public static boolean palindrome(int num) {

    boolean amountDigits = checkDigits(num); //calling our checkDigits method
    if(!amountDigits) {  //shortcut for --> if(amountDigits == false)
        System.out.println("Number must be 3 digits.");
        return false;
    }

    int ones = num%10;
    int hundreds = num/100;

    return ones == hundreds;
}

public static boolean checkDigits(int num) {
    int count = 0;
    while(num != 0) {
        count++;
        num/=10;
    }
    return count == 3;
}
```

SCOPE OF VARIABLES IN METHODS

- Just like with for loops, variables inside our parameters and the body of our methods **only** exist inside our method. They **DO NOT** exist outside of the method.

- **Ex** - s doesn't exist outside of the method:

```
public class Main {

    public static void main(String[] args) {
        printString( s: "hello!!",  amount: 7);
        System.out.println(s);
    }
    public static void printString(String s, int amount) {
        for(int i = 0; i<amount; i++) {
            System.out.println(s);
        }
    }
}
```

Output

```
Error:(9, 28)  java: cannot find symbol
                 symbol:   variable s
                 location: class com.kaitlinHoff.Main
```

REUSING A METHOD NAME

- Do you think you can have two methods with the same name?

REUSING A METHOD NAME

- Do you think you can have two methods with the same name? **It depends!**
- If you have two or more methods all with the same name in the same class, as long as they take a **different number** of parameters or take **different types** of parameters, you can!
- This is called **method overloading**.

REUSING A METHOD NAME

- The first one is **method overloading** and is okay! The second is not allowed since the parameters are identical.

```
public static int sum(int num1, int num2) {
    return num1 + num2;
}
public static double sum(double num1, double num2) {
    return num1 + num2;
}
```

```
public static int sum(int num1, int num2) {
    return num1 + num2;
}
public static double sum(int num1, int num2) {
    return num1 + num2;
}
```

EXAMPLE 1 – METHOD OVERLOADING:

- Write a method that takes in two integers and finds which one has the greatest value.
- Using method overloading, find the greatest value between two doubles.
- Using method overloading, find the greatest value between 3 integers.

EXAMPLE 1 – METHOD OVERLOADING:


```java
public class Main {

    public static void main(String[] args) {

        int maxInt = max(2,5); //calls 1st method
        System.out.println(maxInt);

        double maxDouble = max(4.56, 18.96); //calls 2nd method
        System.out.println(maxDouble);

        int maxIntOfThree = max(7,10,3);
        System.out.println(maxIntOfThree); //calls 3rd method

    }
    //method 1
    public static int max(int i1, int i2) {
        if(i2 > i1) {
            return i2;
        }
        return i1;
    }
    //method 2
    public static double max(double d1, double d2) {
        if(d1 > d2) {
            return d1;
        }
        return d2;
    }
    //method 3
    public static int max(int i1, int i2, int i3) {
        if(i1 > i2 && i1 > i3) {
            return i1;
        } else if(i2 > i1 && i2 > i3) {
            return i2;
        }
        return i3;
    }

}
```

Output:

```
5
18.96
10
```

EXAMPLE 1 – METHOD OVERLOADING:

- Both max(double, double) and max(int, int) are possible matches for max(2, 5). The Java compiler finds the method that **best matches** a method invocation.
- Since the method max(int, int) is a **better match** for max(2, 5) than max(double, double), max(int, int) is used to invoke max(2, 5).

AMBIGUOUS INVOCATION

- Sometimes there are **two or more** possible matches for the invocation of a method, but the compiler **cannot** determine the best match.
- This is referred to as **ambiguous invocation**. Ambiguous invocation causes a compile error. Consider the following code…

Week 5 › src › com › kaitlinHoff › Main

Main.java

```
1  package com.kaitlinHoff;
2
3  public class Main {
4
5      public static void main(String[] args) {
6
7          System.out.println(max(1, 2));
8      }
9
10
11
12
13
14 @    public static double max(int num1, double num2) {
15         if (num1 > num2)
16             return num1;
17         else
18             return num2;
19     }
20 @    public static double max(double num1, int num2) {
21         if (num1 > num2)
22             return num1;
23         else
24             return num2;
25     }
26 }
27
```

Ambiguous method call. Both
max (int, double) in **Main** and
max (double, int) in **Main** match

Main › main()

Messages: Build

Information: java: Errors occurred while compiling module 'Week 5'
Information: javac 11.0.1 was used to compile java sources
Information: 9/29/21 1:35 PM – Compilation completed with 1 error and 0 warnings in 1 s 879 ms
/Users/kaitlinhoffmann/Desktop/CS1/Week 5/src/com/kaitlinHoff/Main.java
Error:(9, 28) java: reference to max is ambiguous
both method max(int,double) in com.kaitlinHoff.Main and method max(double,int) in com.kaitlinHoff.Main match

- Both max(int, double) and max(double, int) are possible candidates to match **max(1, 2)**.
- Because neither is better than the other, the invocation is **ambiguous**, resulting in a compile error.

IS THIS METHOD OVERLOADING?

```
public static void method(int x) {
    System.out.println(x);
}

public static int method(int y) {
    return y+2;
}
```

IS THIS METHOD OVERLOADING? — NO

```
public class Main {

    public static void main(String[] args) {


    }
    public static void method(int x) {
        System.out.println(x);
    }
    public static int method(int y) {
        retu
    }
}
```

'method(int)' is already defined in 'com.kaitlinHoff.Main'

```
es:  Build ×
Information: java: Errors occurred while compiling module 'Week 5'
Information: javac 11.0.1 was used to compile java sources
Information: 9/29/21 1:44 PM – Compilation completed with 1 error and 0 warnings in 1 s 947 ms
/Users/kaitlinhoffmann/Desktop/CS1/Week 5/src/com/kaitlinHoff/Main.java
Error:(12, 23)  java: method method(int) is already defined in class com.kaitlinHoff.Main
```

The variable name doesn't make a difference here. It's the **data type** of the variable that must be different. x and y are **both integers** so this results in an error.

IMPROVING QUALITY OF A PROGRAM

- **Modularization** is a technique to divide a software system into multiple discrete and independent modules (or classes), which are expected to be capable of carrying out tasks independently.
  - ✦ These classes may work as **basic constructs** for the entire software.
  - ✦ Designers tend to design classes such that they can be executed and/or compiled **separately** and **independently**.
- Modularizing makes the code easy to **maintain** and **debug** and enables the code to be **reused**.

IMPROVING QUALITY OF A PROGRAM

- Methods can be used to **reduce** redundant code and enable code reuse.
- Methods can also be used to **modularize code** and improve the quality of the program.

ADVANTAGES OF MODULARIZATION

- Smaller components are easier to maintain
- Program can be divided based on functional aspects
- Desired level of abstraction can be brought in the program
  - **Abstraction** is a simplified version of something technical, such as a function, method, or an object in a program.
  - The goal of "abstracting" data is to reduce complexity by removing unnecessary information.

ADVANTAGES OF MODULARIZATION

- Components with **high cohesion** can be re-used again
  - Cohesion refers to what the class (or module) can do.
  - **Low cohesion** would mean that the class does a great variety of actions - it is broad, unfocused on what it should do.
  - **High cohesion** means that the class is focused on what it should be doing, i.e. only methods relating to the intention of the class.
- Concurrent execution can be made possible
- Desired from security aspect
  - We can make our classes and methods private

AN EXAMPLE...

```java
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a positive integer: ");
        int i = sc.nextInt();

        int fact = findFactorial(i);
        System.out.println("Factorial of " + i + " is " + fact);

    }

    public static int findFactorial(int num) {

        if(num < -1) {
            System.out.println("Number must be positive.");
            return -1;
        }

        int fact = 1;
        for(int i = 1; i<=num; i++) {
            fact*=i;
        }
        return fact;
    }

}
```

AN EXAMPLE...

- By encapsulating the code for obtaining the **factorial** in a method, this program has several advantages:

    1. It **isolates the problem** for computing the factorial from the rest of the code in the main method. Thus, the logic becomes *clear* and the program is *easier* to read.

    2. The errors on computing the factorial are **confined** in the findFactorial method, which *narrows* the scope of debugging.

    3. The findFactorial method now can be **reused** by other programs.

SCANNER CLASS METHODS

- We have been using the Scanner class to read in values from the user.
- Guess what? nextInt(), nextDouble(), next(), etc. are all **methods** from the Scanner class!
  - Notice how the methods above **don't have any parameters**.
- This shows how methods can make our lives easier so we don't have to recreate the same code over and code again.
- The Scanner class and its methods from the Java Docs: https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html

USEFUL METHODS FROM THE MATH CLASS

- So far we've been writing n*n in order to perform exponentiation on a number. However, there is a method already available in the **Math class** if we want to raise n to an even higher power. The following are some useful methods:
  - **Math.pow(a,b)** raises a to power b
  - **Math.PI** is a constant to hold pi
  - **Math.sqrt(a)** returns square root of a
  - **Math.random()** returns a random double from the interval [0,1)
- https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html

USEFUL METHODS FROM THE MATH CLASS

`static double`	**`PI`** The `double` value that is closer than any other to *pi*, the ratio of the circumference of a circle to its diameter.
`static double`	**`pow`**`(double a, double b)` Returns the value of the first argument raised to the power of the second argument.
`static double`	**`random`**`()` Returns a `double` value with a positive sign, greater than or equal to `0.0` and less than `1.0`.
`static double`	**`sqrt`**`(double a)` Returns the correctly rounded positive square root of a `double` value.

USEFUL METHODS FROM THE MATH CLASS

We don't have to create a new object like with Scanner. Instead, we just write ***Math.<method>***

```
double pi = Math.PI;
System.out.println("PI = " + pi);

double power = Math.pow(2,8);
System.out.println("2 raised to the power of 8 = " + power);

double r = Math.random();
System.out.println("Random double between [0,1) = " + r);

double sqrt = Math.sqrt(64);
System.out.println("The square root of 64 = " + sqrt);

System.out.println();
```

Output

```
PI = 3.141592653589793
2 raised to the power of 8 = 256.0
Random double between [0,1) = 0.3399223799905331
The square root of 64 = 8.0
```