Lecture 06

Methods!
Methods

• Methods (sometimes called functions in other languages) help to:

  • Make code reusable. Less typing = less errors = WIN

  • modularize code - group similar tasks together to keep track of where things are happening (great for debugging!)
Challenge: summing ints

- Write a program that will add up all the numbers from 1 to 10
public class SummingInts {

    public static void main(String[] args) {
        int sum = 0;

        for (int i = 1; i <= 10; i++) {
            sum += i;
        }

        System.out.println(sum);
    }
}

Sum 1-10
Change it up

- Sum from 10-20
- Sum from 22-45
- Sum from 1-234, and from 3-56

☝️ is a lot of redundant code!
• Methods help us to reuse code in a sane way.
Methods

• Methods need to be defined before they are called or invoked

• Definition syntax:

```java
modifier(s) returnType methodName(params){
    statement(s);
}
```

• Call or invocation syntax:

```java
methodName(params);
```

or:
```
datatype variableName = methodName(params);
```
Methods

• Example definition:

```java
public static int max(int num1, int num2){
    return result;
}
```

• Example invocation:

```java
int z = max(x, y);
```

• **Note:** For now, all our methods will have ‘static’ as a modifier. We’ll talk about this later, but it’s important to use it for now!
Sum example using methods

- Write a method that takes 2 integers as parameters and adds up all the numbers from the first number to the last

- print the sum of 1-10 in the main method
```java
public class SummingInts_withMethod {

    public static int sum(int num1, int num2)
    {
        int sum = 0;

        for (int i = num1; i <= num2; i++){
            sum += i;
        }

        return sum;
    }

    public static void main(String[] args) {

        int sumOfNumbers = sum(1,10);
        System.out.println(sumOfNumbers);
    }
}
```
Void?

• void means that the method does not return a value.

• Our main methods have been void, because they don’t return anything when they run.

• print methods are good examples of void methods.

• you don’t need a return statement, but you can use it to get out of a method.

• if you want the method to return a result, you must have a return statement in the method.
import java.util.*;

public class voidReturn {

    public static void printUntilUpper(String word){
        String upperWord = "";
        for (int i = 0; i < word.length(); i++){
            if (Character.isUpperCase(word.charAt(i))){
                return;
            } else {
                System.out.print(word.charAt(i));
            }
        }
    }

    public static void main(String args[]){
        Scanner input = new Scanner(System.in);
        String userWord;
        System.out.println("Enter a word: ");
        userWord = input.nextLine();
        printUntilUpper(userWord);
    }
}
Passing arguments

• Arguments or parameters are stuff you can pass into a method to use

• Matching the order of parameters matters

• Matching the number of parameters matters

• Matching compatible types of parameters matters
Passing arguments

• arguments are passed by *value*

• the *value* of the argument is passed, not the actual object
public class PassByValue {

    public static void main(String[] args) {

        int x = 1;
        System.out.println("Before the call: " + x);
        addTen(x);
        System.out.println("After the call: " + x);
    }

    public static void addTen(int num) {
        num = num + 10;
        System.out.println("inside the call: " + num);
    }
}

Overloading methods

- Can have multiple methods with the same name but different parameters
- Useful if you have methods that do similar things, but could have different input
public class OverloadedMethods {

    public static void main(String[] args) {

        System.out.println("The sum is " + sumNums(1.0, 2, 4));
    }

    public static int sumNums(int num1, int num2) {
        int sum = num1 + num2;
        return sum;
    }

    public static double sumNums(double num1, double num2) {
        double sum = num1 + num2;
        return sum;
    }

    public static double sumNums(double num1, double num2, double num3) {
        double sum = sumNums(num1, num2) + num3;
        return sum;
    }

}
Overloading methods

• When there are multiple overloaded methods with the same number of parameters and compatible types, Java will pick the best match for it.

• If there are 2 or more possible matches, you’ll get an error because it’s ambiguous which one is a better match.
• Variables declared inside a method are *local variables*. They are only seen inside the block where they are declared.

• blocks can be a method block, or a loop, for example
public class VariableScope {

    public static void main(String[] args) {

        // Outer loop of "rows"
        for (int i = 0; i < 5; i++) {

            // Inner loop of "columns"
            for (int j = 0; j < 5; j++) {
                // Print *
                System.out.print(j);
            }

            // Go to next "row"
            System.out.println(" ");
        }
    }
}
Scope of variables

```java
public static void method1() {
    for (int i = 1; i < 10; i++) {
        int j;
    }
}
```

The scope of `i`:

The scope of `j`: