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!)
Loop Practice: 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);
    }
}

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

- 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:

  ```java
  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!
Methods

Define a method

```
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

Invoke a method

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

- Many methods return a value, like the `int` in the `max()` method.
- `void` means that the method does not return any 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);
    }

}
Challenge: passing arguments

• Write a program that includes a method that takes 2 parameters - a char, and an int - and prints that character however many times was specified by the int
public class PrintThem {
    public static void main (String[] args) {

        printXTimes('B', 4);

    }

    public static void printXTimes(char c, int times){

        for (int i = 0; i < times; i++){
            System.out.print(c);
        }
    }
}
Overloading methods

• In Java, you 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 and the compiler can’t find the best match, you’ll get an error because it’s ambiguous.
Scope of variables

- 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

- *Variables live and die within the curly braces*
Scope of variables

```java
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("\n");

        }

    }

}
```
Scope of variables

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

• Write a program that asks the user to enter a password given the following rules:

  • it must have at least one letter and one number
  • it must have at least one capital letter and one lowercase
  • it must be at least 8 characters long

• Write three methods, one for each of the rules, and print out “PASS” if it passes the rule and “FAIL” if it fails it for each rule
import java.util.*;
public class PasswordCheck {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Please enter a password: ");
        String password = input.next();

        ruleOne(password);
        ruleTwo(password);
        ruleThree(password);
    }

    public static void ruleOne(String password){
        int letterCount = 0;
        int numberCount = 0;

        for (int i = 0; i < password.length(); i++){
            if (Character.isLetter(password.charAt(i)){
                letterCount++;
            } else if (Character.isDigit(password.charAt(i))){
                numberCount++;
            }
        }

        if (letterCount > 0 && numberCount > 0){
            System.out.println("Rule 1: PASS");
        } else {
            System.out.println("Rule 1: FAIL");
        }
    }

    public static void ruleTwo(String password){
        int upperCount = 0;
        int lowerCount = 0;

        for (int i = 0; i < password.length(); i++){
            if (Character.isUpperCase(password.charAt(i))){
                upperCount++;
            } else if (Character.isLowerCase(password.charAt(i))){
                lowerCount++;
            }
        }

        if (upperCount > 0 && lowerCount > 0){
            System.out.println("Rule 2: PASS");
        } else {
            System.out.println("Rule 2: FAIL");
        }
    }

    public static void ruleThree(String password){
        if (password.length() > 7){
            System.out.println("Rule 3: PASS");
        } else {
            System.out.println("Rule 3: FAIL");
        }
    }
}
More practice

- Problem 6.31 in the book explains the algorithm for validating credit cards - it’s a great example to practice using multiple methods in your programs