Lecture 06

Methods!
But first...

- Let’s take a look at the leap year problem from last class
import java.util.*;

public class FindNextLeapYears {

    public static void main(String[] args) {
        int numberOfLeapYearsToFind;
        int userYear;

        Scanner input = new Scanner(System.in);

        System.out.print("Enter the year you'd like to start from: ");
        userYear = input.nextInt();

        System.out.print("Enter how many leap years you'd like to find: ");
        numberOfLeapYearsToFind = input.nextInt();

        int foundLeapYears = 0;
        while (foundLeapYears < numberOfLeapYearsToFind) {
            if (((userYear % 4 == 0) && (userYear % 100 != 0)) || userYear % 400 == 0) {
                foundLeapYears++;
                System.out.println(userYear);
            }

            userYear ++;
        }
    }
}
Reusing Code

• Loops help

• Another way is creating methods
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);
    }

}
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

- Syntax:

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

• Example definition:

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

• Example invocation:

```java
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
Sum example using methods

```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

- Order matters
- Matching the number of parameters matters
- Matching compatible types matters
Passing arguments

• arguments are passed by *value*

• the *value* of the argument is passed, not the actual object
Passing arguments

```java
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
Overloading methods

```java
public class PrintNumbersOverloaded {

    public static void main(String[] args) {

        printNumbers(10, 3);
    }

    public static void printNumbers(int num1, int num2){
        System.out.println("Number 1: "+ num1);
        System.out.println("Number 2: "+ num2);
    }

    public static void printNumbers(double num1, double num2){
        System.out.println("Number 1: "+ num1);
        System.out.println("Number 2: "+ num2);
    }

    public static void printNumbers(double num1, double num2, double num3){
        System.out.println("Number 1: "+ num1);
        System.out.println("Number 2: "+ num2);
        System.out.println("Number 3: "+ num2);
    }

}
```
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
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(" ");
        }
    }
}
```