1 Introduction

A program
public class Kilograms {
    public static void main (String [] args) {
        final double pounds = 20.0;
        System.out.print(pounds);
        System.out.print(" pounds is ");
        System.out.print(pounds / 2.2);
        System.out.println(" kilograms.");
        return;
    }
}

A program

A program

Java class files

Run exercise in online text

Parts of the program

public class Kilograms {

    public static void main (String [] args) {
        double pounds = 20;
        System.out.print(pounds);

}
Add comments to describe what the program does

/**
 * Converter from pounds to kilograms.
 */
public class Kilograms {
  public static void main (String [] args) {
    double pounds = 20; // Amount to convert to kilograms
    System.out.print(pounds);
    System.out.print(" pounds is ");
    System.out.print(pounds / 2.2);
    System.out.println(" kilograms.");
    return;
  }
}

Input as well as output

import java.util.Scanner; // User input

/**
 * Converter from pounds to kilograms.
 */
public class Kilograms {
  public static void main (String [] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("How many pounds? ");
    final double pounds = scanner.nextDouble();
    final double kilograms = pounds / 2.2;
    System.out.print(pounds);
    System.out.print(" pounds is ");
    System.out.print(kilograms);
    System.out.println(" kilograms.");
    return;
  }
}

The other things that happens with a program
The other things that happen with a program

Errors are frustrating

Good news! There are really only six things you need to know here!

The six things a program can do
1. Get input
2. Give output
3. Do arithmetic
4. Update a stored value
5. Test a condition, and select an alternative
6. Repeat a group of actions

The less good news is that there’s a whole bunch of detail and skill associated with each of these.

Four ways Java will help you organize your work

1. Grouping related data together
2. Defining sequences of operations
3. Associating data with operations relevant to the particular data
4. Naming these groups, sequences and associations for easy and repeated use

2 Simple imperative programming

The six things

1. Get input
2. Give output
3. Do arithmetic
4. Update a stored value
5. Test a condition, and select an alternative
6. Repeat a group of actions

In and out of the system

Where exactly is our program running?

• Should it matter?

• Abstract away from certain details

• There’s some source for input, and some destination for output
  – Give them a name, and describe operations on them
  – But don’t get hung up on the details of exactly what they are
  – Java calls them: System.in and System.out
– Some operations on them are written as a suffix:

```java
System.out.print("Hello...");
System.out.println("again");
```

– For some operations we use a helper:

```java
Scanner scnr = new Scanner(System.in);
// ... then later ...
int quantity = scnr.nextInt();
```

### 2.1 Names and assignments

**Declarations**

- Creates a place in the computer for a value to be stored
  - Give the place a name
  - Specify what type of item goes there
    * Java is *strongly typed* - once we declare a particular type, we have to be consistent
    * So once an integer, always an integer; once a Scanner, always a Scanner
  - Assign an initial value to it

- We saw Tuesday:

```java
Scanner scnr = new Scanner(System.in);
int quantity = scnr.nextInt();
```

- In today’s reading:

```java
int litterSize = 3;
int yearlyLitters = 5;
int annualMice = 0;
```

**Pick good names**

- Use lower camel-case for variable names.
  - Named after the "humps" of upper-case letters in the middle of the name
- Descriptive names, but be reasonable with length
- Use letters, and maybe numbers at the very end
- *Mnemonic* - assisting the memory
  *Consistent* - ease understanding
It’s an update, not an equation

```java
public static void main (String [] args) {
    Scanner scanner = new Scanner(System.in); // 0
    int total = 0; // 1

    System.out.print("Enter a number: "); // 2
    int num = scanner.nextInt(); // 3
    total = total + num; // 4

    System.out.print("Enter another number: "); // 5
    num = scanner.nextInt(); // 6
    total = total + num; // 7

    System.out.print("Just one more number: "); // 8
    num = scanner.nextInt(); // 9
    total = total + num; // 10

    System.out.println("Their sum is " + total); // 11
    return; // 12
}
```

• Step through with inputs 4,7 and 2

Many types of numbers
You’ve seen already:

• int — Rounded integer values
• double — Real-number values

Java also has:

• long — Integer values from a larger range
  - int runs from -2,147,483,648 to 2,147,483,647
  - long runs from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

• byte and short — Integer values from shorter ranges
  - byte runs from -128 to 127
  - short runs from -32,768 to 32,767

• float — Less accurate real-number values
  - There are limits not just in magnitude, but also in accuracy.
  - float runs from about \(-10^{38}\) to \(10^{38}\) with about 7 significant digits of accuracy
  - double runs from about \(-10^{308}\) to \(10^{308}\) with about 16 significant digits of accuracy

Generally:
• Use int or long normally
• Never use a floating-point type when an integer will do
• Only use byte or short to really make a point about the limited range

Clock time

```java
public static void main (String [] args) {
    final Scanner scanner = new Scanner(System.in); // 1
    System.out.print("How many seconds? "); // 2
    int given = scanner.nextInt(); // 3

    final int hours = given / (60 * 60); // 4
    given = given % (60 * 60); // 5
    final int minutes = given / 60; // 6
    final int seconds = given % 60; // 7

    System.out.printf("%d:%02d:%02d\n", hours, minutes, seconds); // 8
    return; // 9
}
```

• Try inputs: 3,923, 2,348 and 3,606.

Printing the clock time

What about the last statement of the clock time program?

`System.out.printf("%d:%02d:%02d\n", hours, minutes, seconds);`

• If you tried the program, you know it just prints the time
• Why is it different than this?

```
    System.out.print(hours);
    System.out.print(":");
    System.out.print(minutes);
    System.out.print(":");
    System.out.println(seconds);
```

– If we print the number 3, would we see 3 or 03?
  `System.out.print(3);`

• `printf` abbreviates `print according to format`
  – Everything in the first argument prints as normal, except for percent signs
  – The percent signs indicate how we should print things
Printing the clock time

- `printf(control_string, value, value, ...);`
  - Only the control string is (necessarily) printed
- `%d` means: take the next argument, print it as an integer (d for decimal)
  - `%2d` means: take two spaces for it
  - `%02d` means: fill up extra space with leading zeroes
- Other codes for strings, floating-point, etc.
  - We’ll come back to `printf` later

2.2 Leap years

Calculating a leap year

- How do we know if a year is a leap year?
  - Easy: if it’s divisible by four
- An algorithm using the Six Things
- In Java:

```java
import java.util.Scanner;

public class LeapYear {
    public static void main(String[] args) {
        final Scanner scnr = new Scanner(System.in);  
        System.out.print("What year? ");  
        final int year = scnr.nextInt();
        if (year % 4 == 0) {
            System.out.println(year + " was a leap year!");
        } else {
            System.out.println(year + " was not a leap year.");
        }
        return;
    }
}
```

- Try 1986 and 1900
Really calculating a leap year

- There are exceptions to the divide-by-four rule
  - Years divisible by 100 are not leap years
  - Unless they are also divisible by 400
- So we need to make multiple decisions

```java
import java.util.Scanner;

public class LeapYear {
    public static void main(String[] args) {
        final Scanner scnr = new Scanner(System.in); // 1
        System.out.print("What year? "); // 2
        final int year = scnr.nextInt(); // 3
        if (year % 400 == 0) { // 4
            System.out.println(year + " was a leap year!"); // 5
        } else if (year % 100 == 0) { // 6
            System.out.println(year + " was not a leap year."); // 7
        } else if (year % 4 == 0) { // 8
            System.out.println(year + " was a leap year!"); // 9
        } else { // 10
            System.out.println(year + " was not a leap year.");
        }
        return; // 11
    }
}

• Try 1986 and 1900

Calculate, then output

```java
import java.util.Scanner;

public class LeapYear {
    public static void main(String[] args) {
        final Scanner scnr = new Scanner(System.in); // 1
        System.out.print("What year? "); // 2
        final int year = scnr.nextInt(); // 3
        final boolean isLeapYear; // 4
        if (year % 400 == 0) { // 5
            isLeapYear = true; // 6
        } else if (year % 100 == 0) { // 7
            isLeapYear = false; // 8
        }
    }
}
import java.util.Scanner;

public class LeapYear {
    public static void main(String[] args) {
        final Scanner scnr = new Scanner(System.in);
        System.out.print("What year? ");
        final int year = scnr.nextInt();

        final boolean isLeapYear;
        if (year % 400 == 0) {
            isLeapYear = true;
        } else if (year % 100 == 0) {
            isLeapYear = false;
        } else if (year % 4 == 0) {
            isLeapYear = true;
        } else {
            isLeapYear = false;
        }

        if (isLeapYear) {
            System.out.println(year + " was a leap year!");
        } else {
            System.out.println(year + " was not a leap year.");
        }

        return;
    }
}

One more tweak

A debugging trick
final boolean isLeapYear;
if (year % 400 == 0) {
    System.out.println("*** branch 1 ***"); // FIXME delete
    isLeapYear = true;
} else if (year % 100 == 0) {
    System.out.println("*** branch 2 ***"); // FIXME delete
    isLeapYear = false;
} else if (year % 4 == 0) {
    System.out.println("*** branch 3 ***"); // FIXME delete
    isLeapYear = true;
} else {
    System.out.println("*** branch 4 ***"); // FIXME delete
    isLeapYear = false;
}

**Boolean operators**

- Conditions are not allowed just in if statements
- Just as there are types for numbers, there is a type for boolean values

    final boolean flag = x<5;

- There are two boolean constants, true and false
- Just as there are operators for integers, there are operators for boolean values

    || or, disjunction
    && and, conjunction
    ! not

**About exceptions**

- Later, we’ll look at code that generates or catches exceptions
- For now, you should just aware of them
  - You will see them as you debug your programs
  - Exercise: use (alter if you need) one of the programs from the book or from a lab exercise to make Eclipse throw an exception.
  - What does it look like?

**2.3 The for-loop**

A simple loop
public class SimpleLoop {
    public static void main(String[] args) {
        for(int i=0; i<10; i++) { // 1
            final int squared = i*i; // 2
            System.out.println(i + " squared is " + squared); // 3
        }
    }
}

How the for-loop works

General loop structure:

for(int VARIABLE=START; CONTINUE_CONDITION; CHANGE) {
    STATEMENT1;
    STATEMENT2;
    ...
    STATEMENTn;
}

• Steps Java takes:
  – int VARIABLE=START
  – Check CONTINUE_CONDITION, maybe stop running the loop
  – Run STATEMENT1 through STATEMENTn
  – Apply the CHANGE
  – Check CONTINUE_CONDITION, maybe stop running the loop
  – Run STATEMENT1 through STATEMENTn
  – Apply the CHANGE
  – Check CONTINUE_CONDITION, maybe stop running the loop
  – Run STATEMENT1 through STATEMENTn
  – Apply the CHANGE
  – Check CONTINUE_CONDITION, maybe stop running the loop
  – Run STATEMENT1 through STATEMENTn
  – Apply the CHANGE
  – ... and so on until the CONTINUE_CONDITION is falsified

2.4 Factorials

What are factorials?
A factorial is the product of all the integers from 1 up to some number.

• The factorial of 7 is $7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$.
• Abbreviated 7!. 
How can we implement the factorial function?

```java
import java.util.Scanner;

public class Factorial {
    public static void main(String[] args) {
        final Scanner scnr = new Scanner(System.in);
        System.out.print("Factorial of: ");
        final int num = scnr.nextInt();

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

        System.out.println("It’s: " + product);
        return;
    }
}
```

A factorial calculator

```java
import java.util.Scanner;

public class Factorial {
    public static void main(String[] args) {
```
final Scanner scnr = new Scanner(System.in); // 1
System.out.print("Factorial of: "); // 2
final int num = scnr.nextInt(); // 3
long product = 1; // 4

for(int n=2; n<=num; n++) { // 5
    product = product * n; // 6
}

System.out.println("It’s: "+ product); // 7
return; // 8
}

A different factorial calculator

import java.util.Scanner;

public class Factorial {
    public static void main(String[] args) {
        final Scanner scnr = new Scanner(System.in);
        System.out.print("Factorial of: ");
        final int num = scnr.nextInt();
        long product = 1;

        for(int factor=num; factor>0; factor--) { // 6
            product = product * factor;
        }

        System.out.println("It’s: "+ product);

        return;
    }
}

• Try it with 4

The n-choose-m function
Factorial shows up in probability and statistics
For example:
• Let’s say we have a bag of $n$ distinct objects
• How many ways are there to choose $m$?
• We write this as $\binom{n}{m}$. 
• It’s calculated as \( \binom{n}{m} = \frac{n!}{m!(n-m)!} \)

**How it works**

**Five things: A B C D E**

How can we choose 3?

• A B C
• A B D
• A B E
• A C D
• A C E
• A D E
• A C D
• A C E
• A D E
• C D E

• There are
  – Five ways to pick one thing out of five
  – Four ways to pick a second one from the remaining four
  – Three ways to pick a third one from the remaining three
  – That’s 5 × 4 × 3

• But we don’t want to consider the order of these three things
  – Just ABC
  – Not ABC, ACB, BAC, BCA, CAB and CBA
  – There are \( n! \) ways to order \( n \) things

• So a first try would be \( \binom{n}{m} = \frac{n(n-1)\cdots(n-m+1)}{m!} \)
  – But the whole ellipses thing is informal and unsatisfying

• But note that we can multiply by \( (n - m)! \) in both the numerator and denominator
  – That makes the top just \( n! \)
  – And the whole formula \( \binom{n}{m} = \frac{n!}{m!(n-m)!} \)

**How would we implement a choose calculator?**
A simple choose calculator

```java
final Scanner scnr = new Scanner(System.in); // 0
System.out.print("How many total objects? "); // 1
final int total = scnr.nextInt(); // 2
System.out.print("Choose many? "); // 3
final int choose = scnr.nextInt(); // 4

long totalFact = 1; // 5
for(int n=2; n<=total; n++) { // 6
    totalFact *= n; // 7
}
for(int n=2; n<=total; n++) { // 8

long chooseFact = 1; // 9
for(int n=2; n<=choose; n++) { // 10
    chooseFact *= n; // 11
}

long diffFact = 1; // 13
for(int n=2; n<=total-choose; n++) { // 14
    diffFact *= n; // 15
}

final long ways = totalFact / chooseFact / diffFact; // 17
System.out.println(ways + " ways to choose"); // 18
return; // 19
```

• Try it with 4, 2

Do less work!

• Note that when $n > m$ we have

$$n! = n \times (n-1) \times \cdots \times (n-m+1) \times (n-m)!$$

• So for $\binom{n}{m}$, we have

$$\binom{n}{m} = \frac{n!}{m!(n-m)!} = \frac{n \times (n-1) \times \cdots \times (n-m+1) \times (n-m)!}{m!(n-m)!}$$

$$= \frac{n \times (n-1) \times \cdots \times (n-m+1)}{m!}$$

• Much easier to calculate!
A better choose calculator

```java
final Scanner scnr = new Scanner(System.in); // 0
System.out.print("How many total objects? "); // 1
final int total = scnr.nextInt(); // 2
System.out.print("Choose many? "); // 3
final int choose = scnr.nextInt(); // 4

long ways = 1; // 5
int totalFactor = total; // 6
for(int i=1; i<=choose; i++) { // 7
    ways = ways * totalFactor / i; // 8
    --totalFactor; // 9
}
System.out.println(ways + " ways to choose"); // 10
return; // 11
```

- Try it with 23, 4

### 2.5 How long is that number?

**How to we find the length in digits of an integer?**

- Use logarithms
  - $1,000 = 10^3$, so $\log_{10} 1,000 = 3$
  - $10,000 = 10^4$, so $\log_{10} 10,000 = 4$

- The number of digits is one more than the $\log_{10}$

- What about non-even powers of 10?
  - $1,000 < 3,162 < 10,000$, so $\log_{10} 1,000 < \log_{10} 3,162 < \log_{10} 10,000$
  - We need to round the logarithm down, then add 1

**The Math class**

- Java has many math functions in its standard library, including `floor` for rounding down, and `log10`

- Its full name is `java.lang.Math`
  - Classes in `java.lang` (unlike `java.util`) do not need an import

- Calculate the number of digits in an integer:

```java
final Scanner scnr = new Scanner(System.in);
System.out.print("Enter a natural number: ");
final int num = scnr.nextInt();
final long digitCount = 1+Math.round(Math.floor(Math.log10(num)));
System.out.println(num + " has " + digitCount + " digits");
return;
```
– Are we rounding twice?
– Why long?
  * Use the Javadoc!
– Always address warnings!

2.6 Fibonacci numbers

What are the Fibonacci numbers

- The Fibonacci sequence starts with 1, then 1, then every subsequent number is the sum of the previous two.
  - 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, · · ·

How would we design a Fibonacci calculator?

- OUTPUT a prompt
- INPUT num, to calculate the num-th Fibonacci number
- UPDATE (INITIALIZE)
  - The current Fibonacci number to 1
  - The previous Fibonacci number to 0
- ITERATE
  - Over values of n from 1 up, while n is less than num
  - ARITHMETIC add the current and previous Fibonacci numbers
  - UPDATE the previous and current Fibonacci numbers
- OUTPUT the current Fibonacci number

A Fibonacci calculator in Java

- OUTPUT a prompt
- INPUT num, to calculate the num-th Fibonacci number
- UPDATE (INITIALIZE)
  - Current to 1
  - Previous to 0
- ITERATE n from 1 up
  - While n < num
  - Add current and previous
  - UPDATE
- OUTPUT current
public class Fibonacci {
    public static void main(String[] args) {
        final Scanner scnr = new Scanner(System.in);
        System.out.print("Which Fibonacci number? ");
        final int num = scnr.nextInt(); // 3
        long thisFib = 1; // Current Fib. num. // 4
        long prevFib = 0; // Previous Fib. num. // 5
        for(int n=1; n<num; n++) { // 6
            final long newFib = thisFib + prevFib; // 7
            prevFib = thisFib; // 8
            thisFib = newFib; // 9
        }
        System.out.println("It's: "+thisFib); // 10
        return; // 11
    }
}

2.7 Fix that sentence

Strings

- We’ve printed strings, but in fact they are values just like numbers and booleans

```java
final String greeting = "hello";
final String name = "Jim";
```

- The built-in operator on strings is *concatenation*, written with the + sign

```java
final String greetingAndName = greeting + name;
System.out.println(greetingAndName);
```

would print out *helloJim* — no space! We must explicitly include a space if we want one

```java
final String greetingAndSpaceAndName = greeting + " " + name;
```

Characters

- Strings are made of characters
- But characters and strings have completely different types!

```java
final char capitalA = 'A';
```

- Strings contains zero or more characters, but a character itself is *exactly* one character
What does this program do?

```java
import java.util.Scanner;

public class FixCaps {
    public static void main(String[] args) {
        final Scanner scnr = new Scanner(System.in); // 1
        System.out.print("Tell me your sentence: "); // 2
        final String sentence = scnr.nextLine(); // 3
        final char firstCharacter = sentence.charAt(0); // 4
        System.out.print(Character.toUpperCase(firstCharacter)); // 5
        for(int i=1; i<sentence.length(); i++) { // 6
            final char thisChar = sentence.charAt(i); // 7
            System.out.print(Character.toLowerCase(thisChar)); // 8
        }
        System.out.println(); // 9
        return; // 10
    }
}
```

• Try it with HELLO!

2.8 Another kind of selection

Characters and switch statements

```java
System.out.print("Give me some letters! ");
final Scanner scnr = new Scanner(System.in);
final String letters = scnr.nextLine();
```
final int len = letters.length();
int vowels = 0;
for(int i=0; i<len; i++) {
    final char letter = letters.charAt(i);
    switch (letter) {
        case 'a':
            vowels += 1;
            break;
        case 'e':
            vowels += 1;
            break;
        case 'i':
            vowels += 1;
            break;
        case 'o':
            vowels += 1;
            break;
        case 'u':
            vowels += 1;
            break;
    }
}
System.out.println("There were " + vowels + " vowels there.");

Many labels

System.out.print("Give me some letters!");
final Scanner scnr = new Scanner(System.in);
final String letters = scnr.nextLine();
scnr.close();

final int len = letters.length();
int vowels = 0;
for(int i=0; i<len; i++) {
    final char letter = letters.charAt(i);
    switch (letter) {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
            vowels += 1;
            break;
    }
}
2.9 Another kind of loop

Fix that sentence again
final Scanner scnr = new Scanner(System.in); // 1
System.out.print("Tell me your sentence: "); // 2
final String sentence = scnr.nextLine(); // 3
int i=0; // 4
while (i<sentence.length()) { // 5
    final char thisChar = sentence.charAt(i); // 6
    if (i == 0) { // 7
        System.out.print(Character.toUpperCase(thisChar)); // 8
    } else { // 9
        System.out.print(Character.toLowerCase(thisChar)); // 9
    }
    i++; // 10
}
System.out.println(); // 11
return; // 12

2.10 Multiple loops

What does this program do?

for (int i=1; i<=6; i++) { // 1
    for (int j=1; j<=i; j++) { // 2
        System.out.print(i); // 3
    }
    System.out.println(); // 4
}

Pitfalls with loop variables

• In a test

for (int i=1; i<=6; i++) { // 1
    for (int j=1; i<=i; j++) { // 2
        System.out.print(i); // 3
    }
    System.out.println(); // 4
}

• In the post-loop effect

for (int i=1; i<=6; i++) { // 5
    for (int j=1; j<=i; i++) { // 6
        System.out.print(i); // 7
    }
    System.out.println(); // 8
}
Two inner loops

```java
for (int i=1; i<=6; i++) {   // 1
    for (int j=1; j<=(6-i); j++) {   // 2
        System.out.print("-");   // 3
    }
    for (int j=1; j<=i; j++) {   // 4
        System.out.print(i);   // 5
    }
    System.out.println();   // 6
}
```

• What if we swap the two inner loops?

Commenting on loops

Document your loop with a high-level comment on the purpose of the loop, but don’t just repeat what the code already says.

• **Good**

```java
// Print a triangle of numbers, each row
// repeating the digit one greater than the row above.
for (int i=1; i<=6; i++) {   // 1
    for (int j=1; j<=i; j++) {   // 2
        System.out.print(i);   // 3
    }
    System.out.println();   // 4
}
```

• **Bad**

```java
// Loop i from 1 to 6, each time loop j from 1 to i, each time print i, and then in the // outer loop start a new line.
for (int i=1; i<=6; i++) {   // 1
    for (int j=1; j<=i; j++) {   // 2
        System.out.print(i);   // 3
    }
    System.out.println();   // 4
}
```

Another triangle

For example, for 5:

```
0
01
012
0123
```
• How can we generate this triangle?

**Another triangle**

For example, for 5:

```java
final Scanner scnr = new Scanner(System.in);
System.out.print("Triangle size: ");
final int num = scnr.nextInt();
scnr.close();

for(int i=0; i<=num; i++) {
    for(int j=0; j<=i; j++) {
        System.out.print(j);
    }
    System.out.println();
}

for(int i=num-1; i>=0; i--) {
    for(int j=0; j<=i; j++) {
        System.out.print(j);
    }
    System.out.println();
}
```

• Can we do this with one loop?

What function will give the number of columns in these rows?

• Function graph

```
|   + 5     |   *
|   /     / |
| + 4    * * |
|   /     / |
| + 3    * * |
|   /     / |
| + 2    * * |
```
• Absolute value!
  – 5-|5-|5
• In Java, it’s Math.abs

One loop

```java
public class SideTriangle {
    public static void main(String[] arg) {
        for(int i=0; i<=10; i++) {
            for(int j=0; j<=5-Math.abs(5-i); j++) {
                System.out.print(j);
            }
            System.out.println();
        }
    }
}
```

Another pattern

```
+ * * *
|/ \
*--------*---------*--
0 1 2 3 4 5 6 7 8 9 10
```

• How can we draw this pattern (in one pair of nested loops)?
  – Overall size of square
  – Size of border

Another pattern

```java
final Scanner scnr = new Scanner(System.in);
System.out.print("Overall size: ");
final int length = scnr.nextInt();
System.out.print("Border thickness: ");
```
final int border = scnr.nextInt();
final int upperBorder = length-border;
scnr.close();

for(int i=0; i<length; i++) {
    for(int j=0; j<length; j++) {
        if (i<border || i>=upperBorder
          || j<border || j>=upperBorder) {
            System.out.print("*");
        } else {
            System.out.print(".");
        }
    }
    System.out.println();
}

3 Methods

So what now?

• We’ve seen all of the Six Things a Program Can Do in action
• Now we move on to a way Java helps us organize
  1. Grouping sequences of operations together
  2. Grouping related data together
  3. Associating a group of data with operations relevant to that data
  4. Naming these groups and associations for easy and repeated use

3.1 Method basics

Remember the Fibonacci calculator

import java.util.Scanner;

public class Fibonacci {
    public static void main(String[] args) {
        final Scanner scnr = new Scanner(System.in); // 1
        System.out.print("Which Fibonacci number? "); // 2
        final int idx = scnr.nextInt(); // 3
        long thisFib = 1; // 4
        long prevFib = 0; // 5

        for(int n=1; n<idx; n++) { // 6
            final long newFib = thisFib + prevFib; // 7
            prevFib = thisFib; // 8
            thisFib = newFib; // 9
        }
    }
}
A Fibonacci calculator method

```java
import java.util.Scanner;

public class Fibonacci {
    public static void main(String[] args) {
        final Scanner scnr = new Scanner(System.in); // 1
        System.out.print("Which Fibonacci number? "); // 2
        final int idx = scnr.nextInt(); // 3
        final long fib = getFibonacci(idx); // 4
        System.out.println("It’s: " + fib); // 5
        return; // 6
    }

    public static long getFibonacci(int i) { // 7
        long thisFib = 1; // 8
        long prevFib = 0; // 9

        for(int n=1; n<i; n++) { // 10
            final long newFib = thisFib + prevFib; // 11
            prevFib = thisFib; // 12
            thisFib = newFib; // 13
        }

        return thisFib; // 14
    }
}
```

Elements of a method

- **A name** — `getFibonacci`
- **Parameters** — `int i`
  - Any number, zero or more
  - Where we define the method, we have *formal parameters*
    ```java
    public static long getFibonacci(int i) {
    }
    ```
  - Where we use the method, we have *actual parameters*
    ```java
    final long fib = getFibonacci(idx);
    ```
  - *Argument* is a synonym for *parameter*
• A return type — long
• A return value — return thisFib;
  – The return type can be void, in which case there’s no return value

Why do we create a method
• Break up the program into manageable units
• Reuse code from program to program
  – For example, the various Math methods
• Separate different aspects of the program
  – For example, separate user interaction from calculations

Names are local
• The way a name is used is local to methods
• If we assign something else to a name, the effect does not travel outside of the method

```java
public static void main(String[] args) {
    int y = 40;
    f(y);
    System.out.println(y);
}

public static void f(int x) {
    x = 5; // Applies only to f’s variable x
    return;
}
```

3.2 A method for leap years

Reaching way back — calculating a leap year

```java
import java.util.Scanner;

public class LeapYear {
    public static void main(String[] args) {
        final Scanner scnr = new Scanner(System.in);
        System.out.print("What year? ");
        final int year = scnr.nextInt();

        if (year % 400 == 0) {
            System.out.println(year + " was a leap year!");
        } else if (year % 100 == 0) {
            System.out.println(year + " was not a leap year.");
        }
    }
}
```
else if (year % 4 == 0) {
    System.out.println(year + " was a leap year!");
} else {
    System.out.println(year + " was not a leap year.");
}

return;
}

Note that we need the else statements to keep from printing multiple messages

### A leap year method

```java
public static boolean isLeapYear(int year) {
    if (year % 400 == 0) { return true; }
    if (year % 100 == 0) { return false; }
    if (year % 4 == 0) { return true; }
    return false;
}
```

```java
public static void main(String[] args) {
    final Scanner scnr = new Scanner(System.in);
    System.out.print("What year? ");
    final int year = scnr.nextInt();
    final boolean isLeap = isLeapYear(year);
    if (isLeap) {
        System.out.println(year + " was a leap year!");
    } else {
        System.out.println(year + " was not a leap year.");
    }

    return;
}
```

Since return alters the flow of control, we don’t need the else’s here

### 3.3 A choose method

#### The choose calculator

How would we write a method for the n-choose-m function?

```java
public class Choose {
    public static void main(String[] args) {
        final Scanner scnr = new Scanner(System.in);
        System.out.print("How many total objects? ");
        final int total = scnr.nextInt();
        System.out.print("Choose how many? ");
```
A choose method

public class Choose {
    public static long nChooseM(int total, int choose) {
        long ways = 1;
        int totalFactor = total;
        for(int i=1; i<=choose; i++) {
            ways = ways * totalFactor / i;
            --totalFactor;
        }
        return ways;
    }
}

Length methods

How would we write a method characterLength for the length of an integer in digits?

public static int characterLength(long num) {
    return 1+Math.floor(Math.log10(num));
}

public static int characterLength(String str) {
    return str.length();
}
3.4 Factorial and Fibonacci methods

Reaching back again

```java
import java.util.Scanner;

public class Factorial {
    public static void main(String[] args) {
        final Scanner scnr = new Scanner(System.in);
        System.out.print("Factorial of: ");
        final int num = scnr.nextInt();
        long product = 1;

        for(int n=2; n<=num; n++) {
            product *= n;
        }

        System.out.println("It's: "+ product);

        return;
    }
}

We make everything into a method now

```java
public static long factorial(int num) {
    long product = 1;

    for(int n=2; n<=num; n++) {
        product *= n;
    }

    return product;
}

The recursive way

```java
public static long factorial(int num) {
    if (num <= 1) {
        return 1;
    } else {
        return num * factorial(num-1);
    }
}

A first idea for recursive Fibonacci
```
public static long fib(int num) {
    if (num <= 0) {
        return 0;
    } else if (num == 1) {
        return 1;
    } else {
        return fib(num-1) + fib(num-2);
    }
}

Running the first idea
In the recursive implementation for computing the $n$th Fibonacci number with $n = 4$, how many recursive calls are made?

Calls we'd make
fib(5) calls first fib(3), then fib(4)

• fib(3) calls first fib(1), then fib(2)
  – fib(1) is a base case
  – fib(2) calls first fib(0), then fib(1)
    * fib(0) is a base case
    * fib(1) is a base case
• fib(4) calls first fib(2), then fib(3)
  – fib(2) calls first fib(0), then fib(1)
    * fib(0) is a base case
    * fib(1) is a base case
  – fib(3) calls first fib(1), then fib(2)
    * fib(1) is a base case
    * fib(2) calls first fib(0), then fib(1)
    * fib(0) is a base case
    * fib(1) is a base case
That’s 15 calls

• Lots of repeated work!
• Is recursion a bad approach for Fibonacci?
  – Or is this way of structuring recursion bad?
Fibonacci with a loop

```java
public static long fib(final int num) {
    long fib=0;
    long next=1;

    for(int n=0; n<num; n++) {
        final long
        newNext=fib+next;
        fib = next;
        next = newNext;
    }
    return thisFib;
}
```

• The loop runs (about) num times, so we should be able to find a way that recurs num times

• Note how we use next and fib
  – Both calculated on each pass through the loop
  – Both preserved from one pass through the loop to the next

• How can we provide both from one recursive call to the next?
  – By passing both as parameters!

Recursion with accumulating parameters

• Instead of calculating the result in a method body after the return of a recursive call,

• Calculate the result in the arguments of the call

```java
static long fibHelper(int n, long fib, long next) {
    if (n<1) {
        return fib;
    } else {
        return fibHelper(n-1, next, fib+next);
    }
}

public static long fib(int n) {
    return fibHelper(n, 0, 1);
}
```