1 Introduction

A program

```java
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

Inputs

Outputs

Machine code

Run on machine

Compiler
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

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;
    }
}

Parts of the program

/*
 * 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;
    }
}
System.out.print(pounds);
System.out.print(" pounds is ");
System.out.print(pounds / 2.2);
System.out.println(" kilograms."); // End of this line
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);

        // Read a number of pounds, and convert it to kilograms
        System.out.print("How many pounds? ");
        final double pounds = scanner.nextDouble();
        final double kilograms = pounds / 2.2;

        // Print a message about the conversion
        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

public class Kilograms {
    public static void main (String [] args) {
        double pounds = 20;
        System.out.print(pounds);
        System.out.print(" pounds is ");
        System.out.print(pounds / 2.2);
        System.out.println(" kilograms.");

        return;
    }
}

Compiler says we have an error
Nothing!

The other things that happens with a program
public class Kilograms {
    public static void main (String [] args) {
        double pounds = 20;
        System.out.print(pounds);
        System.out.print(" pounds is ");
        System.out.print(pounds / 0.0);
        System.out.println(" kilograms.");
        return;
    }
}

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:
    System.out.print("Hello...");
    System.out.println("again");
  – For some operations we use a helper:
    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
• 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
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); // 1
        System.out.print("What year? "); // 2
        final int year = scnr.nextInt(); // 3

        if (year % 4 == 0) { // 4
            System.out.println(year + " was a leap year!"); // 5
        } else { // 6
            System.out.println(year + " was not a leap year.");
        }

        return; // 7
    }
}
```

- 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 { // 6
            System.out.println(year + " was not a leap year.");
        }

        return; // 7
    }
}
```
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