1 Lecture 1

1.1 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;
    }
}
```
**Parts of the program**

```java
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;
    }
}
```

**Add comments to describe what the program does**

```java
//**
// * 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."); // End of this line
        return;
    }
}
```

**Input as well as output**

```java
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;
    }
}
```
Errors are frustrating

What is the most used language in programming?

Profanity

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

Only six things!

• If that’s all a computer can do, maybe that’s all we have to do this semester??
• There’s a whole bunch of detail and skill associated with each of these
• We will see common patterns of combining the Six Things
• There are also questions about organizing our programs and data…
Four ways Java will help you organize your work

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

Exercise 1.1. In Eclipse, create a new project called Lab 1 - Hello World, and within it create a class called HelloWorld (note that there are spaces in the project name, but there are not spaces in the class name). Place the following import declaration at the beginning of the file, above the start of the class definition (public class HelloWorld...)

```java
import java.util.Scanner;
```

Create a main method for class HelloWorld, or fill in the main routine if it creates one for you, like this:

```java
public class HelloWorld {
    public static void main(String[] args) {
        final Scanner scan = new Scanner(System.in);
        System.out.print("What is your name? ");
        final String name = scan.next();
        System.out.println("Hello " + name + "! Welcome to Java!");
        scan.close();
    }
}
```


2 Lecture 2

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 in the first lecture:
  
  ```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
class Main {  
  public static void main(String[] args) {  
    Scanner scanner = new Scanner(System.in);  
    int total = 0;  

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

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

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

    System.out.println("Their sum is "+total);  
  }  
}
```

Exercise 2.1. Step through this program 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

Operations on numbers

• Usual arithmetic: +, -, *, /
  – But notice that integer division may not be what you expect!
  – Another basic operator: modulus %

• Many other mathematical function are provided as methods in a standard Java library
  – Its full name is java.lang.Math
    * Classes in java.lang (unlike java.util) do not need an import
  – Documentation is online: https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html

Numeric operations example

```java
public class NumOpsExample {
    public static void main (String [] args) {
        final Scanner scanner = new Scanner(System.in); // 1
        System.out.print("Enter a number: "); // 2
        int given = scanner.nextInt(); // 3
        final int doubled = 2*given; // 4
        System.out.println(doubled); // 5
        return; // 6
    }
}
```

Exercise 2.2. Answer these questions by writing short Java programs (or start from the NumOpsExample above, and make changes):

• Does subtraction group to the left, or to the right? That is, when we ask Java to evaluate $100-50-10$, will it evaluate $(100-50)-10$, or will it evaluate $100-(50-10)$?

• Does division group to the left, or to the right?

• Does modulus group to the left, or to the right?

• Does multiplication take precedence over addition, as it does in school algebra? That is, when we ask Java to evaluate $100+50*10$, will it evaluate $(100+50)*10$, or will it evaluate $100+(50*10)$?
Exercise 2.3. Answer the following questions using the java.lang.Math documentation

- What methods does Java provide for logarithms?
- What methods does Java provide for trigonometry?
- What is the difference between `floor` and `ceil`?
- What is the difference between `floor` and `round`?
- What do the `signum` methods do? Why are there two of them?

Exercise 2.4. Write a class `ClockTime` whose `main` method reads in a number of seconds, and prints that length of time as a number of hours, minutes and seconds written with a colon between them in the way we usually write clock times.

For example, for the input 5025 the program should output 1:23:45.

What happens when you run your program on input 7260? Is the output what you would write (or expect to read) for a clock time? If not, why not? We will come back later to this program and fix this problem.

Exercise 2.5. A TwoMult sequence is a sequence of numbers where each number (after the first two) is the product of the two prior numbers. Write a Java class `TwoMult` whose `main` method reads the first two numbers of a TwoMult sequence from the user, and prints the next three numbers of the sequence.

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;
```

Converting numbers into strings

So we have two meanings for + in Java

- It denotes adding numbers
- It also denotes concatenating strings
- We say that addition is an `overloaded` operator

So what if we write + between a number and a string?
final String name = "Jim";
final int number = 2000;
System.out.println(name + number);

In this case, Java assumes that we want to convert the number into a String.

- We could also write

    final String name = "Jim";
    final int number = 2000;
    final String combo = name + number;
    System.out.println(combo);

Example of converting numbers into strings

We could print a message along with the doubled number.

```java
public class NumOpsExample {
    public static void main (String [] args) {
        final Scanner scanner = new Scanner(System.in); // 1
        System.out.print("Enter a number: "); // 2
        int given = scanner.nextInt(); // 3
        final int doubled = 2*given; // 4
        System.out.println("Twice "+ given + " is "+ doubled); // 5
        return; // 6
    }
}
```

3 Lecture 3

PITFALL: 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 3.1. Use (alter if you need) one of the programs from the book or an exercise to make Eclipse throw an exception. What does it look like?

REVIEW: Four ways Java will help you organize your work

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

• They have a name, and we can refer to them by name
• We can provide values to a method, and the method’s operations can use those values
  – Called parameters
  – Each of these values has a definite, specific type
• The method can provide a return value representing the result of its work
  – Also has a definite, specific type
  – Or, it should be declared void if it does not return a value

Specifying methods
For example

```java
public static double
toCelsius(double degreesFahrenheit) {
    return (degreesFahrenheit - 32.0) * 5.0 / 9.0;
}
```

• Named toCelsius
• One parameter, of type double
• Returns a double value
  – The declared type and the type of the value with the return agree

Where do we find methods?

• We can define our own methods
  – Like toCelsius
• Java provides many standard methods for us
  – Like the Math methods we saw last week

Exercise 3.2. Write static methods f1, f2 and so on implementing the following mathematical functions on real numbers (double). Do not use methods from the Math class for these.

1. \( f_1(x) = 2x + 1 \)
2. \( f_2(x, y) = x^2 + 2xy + y^2 \)
3. \( f_3(u) = u^3 + 2u^2 - 3u + 10 \)
4. \( f_4(w) = \frac{w+1}{w-1} \)
5. \( f_5(z) = f_3(z) + f_4(2 + z^2) \)

What happens when we call \( f_4(1) \) from a main method?
Exercise 3.3. Write static methods \( g_1, g_2 \) and so on implementing the following mathematical functions on real numbers (double). Do use methods from the Math class for these.

1. \( g_1(x) = \sqrt{2x^2 + 1} \)
2. \( g_2(x, y) = \log_x y \)
3. \( g_3(w) = |w + 10| \)
4. \( g_4(z) = z^{200} \)

What happens when we call \( f_4(1) \) from a main method?

Exercise 3.4. Convert your clock-time program from Exercise 2.4 into a method. Give your method the name getClockTime. It should take an integer number of seconds, and return a String representing the clock time.

Exercise 3.5. Convert part of the main method of your TwoMult class from Exercise 2.5 into a method printTwoMult. It should take the first two numbers of a TwoMult sequence, and print the next three numbers of the sequence. Your main method should read in two integers as before, but should make a call to printTwoMult instead of calculating and printing the numbers itself.

Exercise 3.6. Most most cereals are made primarily of flour, sugar and high-fructose corn syrup. Write a class CerealMaker with a static method announceComposition. Your method should take three integer arguments, representing (respectively) the number of grams of flour, sugar and high-fructose corn syrup in a standard serving of some particular cereal. Your announceComposition method should print a well-formatted announcement of the total number of grams in a standard serving, repeat the number of grams and the name of each ingredient, and then print the total percentage of the standard serving which is sweetener. Your announceComposition method should not return any result.

Exercise 3.7. Starting with your class ClockTime from Exercise 3.4, discard any main method you may have kept from Exercise 2.4 so that your class contains only the getClockTime method. Add a new main method which validates your method’s behavior. Your method should print several lines of the form

For 5025, expected "1:23:45", got "1:23:45"

where the first string is literally written out in your println statement, and the second string is what your getClockTime method returns.

Exercise 3.8. You have probably run across the factorial function in your math classes. It is defined by two rules:

\[
0! = 0 \\
n! = n \cdot (n - 1)! \quad \text{when } n > 0
\]

We have not yet learned enough Java to implement a factorial method. But we can get ready for when we implement factorial, by writing methods to test our implementation. Notice the difference with Exercise 3.7 — in the previous one we checked what a method was already doing; now, we are setting expectations for what a method will do. This approach is called test-driven development — we write tests first, so that our goals are clear, and so that we can know when our method is correct.

We stub the factorial method by writing an implementation which we know is wrong, but which will compile and run with our tests. By making our tests compile and run (albeit with incorrect results), when we do develop the factorial method, we can do so without worrying that our test infrastructure is lacking.

```java
class FactorialTester {
    public long factorial(int n) {
        // TODO --- later we will implement factorial correctly
        return -1;
    }
}
```

So starting from the above class, add a main method which tests factorial on several different values.
4 Lecture 4

REVIEW: From the book

You’ve read about how Java allows us to describe the fifth Thing that a Computer Can Do: selection

if (CONDITION) {
    // SOME STATEMENTS
} else {
    // SOME OTHER STATEMENTS

} Today we will look at some examples of using if-statements

Exercise 4.1. Write a class Grader with static method getLetterGrade which takes an integer argument representing to a percentage grade from 0 to 100, and returns a string representing the corresponding letter grade,

<table>
<thead>
<tr>
<th>Numeric</th>
<th>Letter</th>
</tr>
</thead>
<tbody>
<tr>
<td>95 ≤ g</td>
<td>A</td>
</tr>
<tr>
<td>92 ≤ g &lt; 95</td>
<td>AB</td>
</tr>
<tr>
<td>86 ≤ g &lt; 92</td>
<td>B</td>
</tr>
<tr>
<td>82 ≤ g &lt; 86</td>
<td>BC</td>
</tr>
<tr>
<td>73 ≤ g &lt; 82</td>
<td>C</td>
</tr>
<tr>
<td>60 ≤ g &lt; 73</td>
<td>D</td>
</tr>
<tr>
<td>g &lt; 60</td>
<td>F</td>
</tr>
</tbody>
</table>

As a first step, write a main method with examples and expected grade calculations.

Exercise 4.2. WidgetCo manufactures several different kinds of widgets for re-sale by various vendors. Based on past relationships, sales targets, and other factors, certain vendors are given discount codes which entitle them to a particular discount on their purchases. Write a class WidgetCoDiscounts with a static method getDiscountedPrice which takes two arguments, a string discount code and an integer base purchase price, and returns the price which should be charged given the particular discount code.

<table>
<thead>
<tr>
<th>Code</th>
<th>Discount</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>10%</td>
</tr>
<tr>
<td>R</td>
<td>12%</td>
</tr>
<tr>
<td>T</td>
<td>15%</td>
</tr>
<tr>
<td>M</td>
<td>3%</td>
</tr>
<tr>
<td>E</td>
<td>8%</td>
</tr>
</tbody>
</table>

If the discount code is an empty string or does not match any in the above table, then the method should return the original base purchase price.

As a first step, write a main method with examples and expected price calculations, for example

System.out.println("For code S and purchase $100, expect 90, got "+ getDiscountedPrice("S", 100));

Remember that we compare strings with .equals, but compare numbers with == .

Exercise 4.3. Square Deal Credit Union offers a program for first-time home buyers to save on the downpayment required for their loan. The downpayment is calculated according to the following table:

<table>
<thead>
<tr>
<th>Purchase price of home</th>
<th>Downpayment required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Under $50,000</td>
<td>4% of price</td>
</tr>
<tr>
<td>$50,000-$124,999</td>
<td>$2,000 plus 8% of price over $50,000</td>
</tr>
<tr>
<td>$125,000-$175,000</td>
<td>$8,000 plus 12% of price over $125,000</td>
</tr>
<tr>
<td>Over $175,000</td>
<td>Not eligible for this program</td>
</tr>
</tbody>
</table>
Write a class `SquareDeal` with static method `getProgramDownpayment` which takes one integer argument representing the home purchase price, and returns an integer representing the required downpayment under this program, or -1 if either the home is not eligible or a negative price is entered. As a first step, write a `main` method with examples and expected downpayment calculations.

**Exercise 4.4.** Write a class `ThreeSorter` with static method `printInOrder` which

- Takes three arguments of type `int`,
- Returns nothing, and
- Prints the three numbers in ascending numeric order.

As a first step, write a `main` method with examples and expected output.

**Exercise 4.5.** The Wisconsin Wants Walnuts company buys black walnuts from individuals, and shells them for sale to restaurants. Sellers’ walnuts are loaded into baskets which hold about one pound of unshelled nuts, and WWW pays $0.15 for each full basket of nuts. If the final, partially-filled basket is more than half-full, WWW pays the full $0.15 for that basket; otherwise they pay $0.05 for the partial basket. Write a class `WalnutBuyer` with a static method `getPurchaseOffer` which

- Takes a single `double` representing the number of baskets of black walnuts brought by a seller (so for example, 3.25 represents three full baskets and an additional basket which is one-quarter full), and
- Returns the amount that WWW will pay for those baskets.

As a first step, write a `main` method with examples and expected payments.

## 5 Lecture 5

**Boolean operators**

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

```java
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

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

**Exercise 5.1.** Write a class `ParityChecker` and static method `isOdd` which

- Takes one parameter of type `int`, and
- Returns a result of type `boolean` which is `true` exactly when the argument is odd.

Do not use any of the methods in the `Math` class for your method. As a first step, write a `main` method with examples and expected results.
Exercise 5.2. Write a class `EvenSquares` and two static methods `isEvenAndSquare` and `isEvenOrSquare` where

- Both methods take one parameter of type `int` and have a result of type `boolean`,
- `isEvenAndSquare` returns `true` when the argument is *both* an even number and a perfect square, and
- `isEvenOrSquare` returns `true` when the argument is *either* an even number or a perfect square.

You are free to use any of the methods in the `Math` class for this exercise. As a first step, write a `main` method with examples and expected results for both methods.

Exercise 5.3. Consider three sticks of length two inches, three inches and six inches. We could not form a triangle with these sticks, because one stick is longer than the other two put together. But if instead the sticks had lengths two inches, three inches and four inches, we could make a triangle from those sticks. Write a class `TriangleLengthsChecker` and static method `isTrianglePossible` which

- Takes three arguments of type `int`,
- Returns a result of type `boolean` which is `true` exactly when sticks of the three lengths could form a triangle.

If one of the lengths is zero or negative, your method should return `false`. As a first step, write a `main` method with examples and expected results.

Exercise 5.4. Write a class `ThreeSorter` with static method `printInOrder` which

- Takes three arguments of type `int`,
- Returns nothing, and
- Prints the three numbers in ascending numeric order.

As a first step, write a `main` method with examples and expected output.

Exercise 5.5. Write a class `LeapYearChecker` with a static method `isLeapYear` which determines whether a year is a leap year. The rules and exceptions for determining whether a year is a leap year are:

- Most years are *not* leap years
- Unless the year is divisible by 4, in which case it *is* a leap year
- Unless the year is also divisible by 100, in which case it is *not* a leap year
- Unless the year is also divisible by 400, in which case it *is* a leap year

Your `isLeapYear` method should take a single argument of type `int` representing the year being tested, and should return its answer as a `boolean`, with `true` denoting a leap year.

6 Lecture 6

PITFALL: Keep things simple

Did your leap year method end this way?

```java
} else if (year % 4 == 0) {
    return true;
} else {
    return false;
}
```
This ending is correct — but it is more complicated than it needs to be

• When \( \text{year} \ % \ 4 == 0 \) evaluates to \text{true}, the method returns \text{true}
• When \( \text{year} \ % \ 4 == 0 \) evaluates to \text{false}, the method returns \text{false}
• Simplify by simply returning \( \text{year} \ % \ 4 == 0 \) itself!

Because simpler methods are

• Easier to understand
• Easier to debug
• Easier to maintain

**REVIEW: From the book**
You’ve read about how Java allows us to describe the sixth Thing that a Computer Can Do: *iteration*

```java
for (int VARIABLE = START; CONTINUATION_CONDITION; CHANGE) {
    STATEMENT1;
    STATEMENT2;
    ...
    STATEMENTn;
}
```

Today we will look at some examples of using for-loops

**How the for-loop works**
General loop structure:

```java
for (int VARIABLE = START; CONTINUATION_CONDITION; CHANGE) {
    STATEMENT1;
    STATEMENT2;
    ...
    STATEMENTn;
}
```

• Steps Java takes:
  - \text{int VARIABLE=START}
  - Check \text{CONTINUATION\_CONDITION}, maybe stop running the loop
  - Run \text{STATEMENT1 through STATEMENTn}
  - Apply the \text{CHANGE}
  - Check \text{CONTINUATION\_CONDITION}, maybe stop running the loop
  - Run \text{STATEMENT1 through STATEMENTn}
  - Apply the \text{CHANGE}
  - Check \text{CONTINUATION\_CONDITION}, maybe stop running the loop
  - Run \text{STATEMENT1 through STATEMENTn}
  - Apply the \text{CHANGE}
  - Check \text{CONTINUATION\_CONDITION}, maybe stop running the loop
  - Run \text{STATEMENT1 through STATEMENTn}
  - Apply the \text{CHANGE}
  - ... and so on until the \text{CONTINUATION\_CONDITION} is falsified
Exercise 6.1. Write a class SimpleLoop whose main method prints the squares of the integers from 0 to 10.

Exercise 6.2. Write a class SentenceFixer with a static method printCapitalized which

- Accepts a String parameter assumed to be a sentence,
- Returns nothing, and
- Prints that sentence making sure the first character is capitalized, and that subsequent characters are lower-case.

The standard methods toUpperCase and toUpperCase in class java.lang.Character will be helpful in converting characters to the correct case. As the usual first step, write a main method with examples and expected results. Step through your method by hand for the argument string HELLO! to be sure you understand you it works.

Exercise 6.3. The factorial function $n!$ is defined informally as $n! = n \cdot (n - 1) \cdot \ldots \cdot 2 \cdot 1$, and is defined formally by two rules:

- If $n = 0$, then $n! = 1$
- If $n > 0$, then $n! = n \cdot (n - 1)!$

Write a class FactorialFinder with a static method factorial which

- Accepts a single int parameter
- Returns a long result representing the factorial of the argument.

Since factorial is not defined on negative numbers, it does not matter what your method does for such input. As the usual first step, write a main method with examples and expected results.

Exercise 6.4. The choose function from probability is defined as

$${n \choose m} = \frac{n!}{m!(n - m)!}.$$ 

Given the factorial method above, it is certainly possible to extend the FactorialFinder of Exercise 6.3 class with a method to implement choose directly:

```java
public static long nChooseM(final int n, final int m) {
    return factorial(n)/factorial(m)/factorial(n-m);
}
```

But this implementation is inefficient, and may cause overflow even when the final result actually can be represented as a long. Write a more efficient version of nChooseM which only performs the multiplications and divisions which are absolutely necessary. As usual, extend the main method with examples and expected results as a first step.

Exercise 6.5. The Fibonacci numbers are a sequence of integers indexed from 0 up, defined by:

- Fibonacci number 0 is 0.
- Fibonacci number 1 is 1.
- For any $n > 1$, Fibonacci number $n$ is the sum of the two previous Fibonacci numbers (indexed $n - 1$ and $n - 2$).

Write a class FibonacciFinder with a static method fibonacci which

- Accepts a single int parameter $n$
- Returns a long result representing Fibonacci number $n$.

Since the series is not defined on negative numbers, it does not matter what your method does for such input. As the usual first step, write a main method with examples and expected results.
Exercise 6.6. Write a class VowelCounter with static method getVowelCount whose one argument is a String and which returns the number of characters in the string which are vowels (a, e, i, o and u).

7 Lecture 7

LOOK BACK: Two looping patterns
We’ve seen two different patterns for combining looping with other Thing a Computer Can Do

Processing each element
• Some action for each value
• But we do not link different values together
• So far, this has been printing

Accumulating a new value
• No action for values by themselves
• But combine them (or something about them) together
• Factorial, Fibonacci
• Typified by an accumulator variable
  – Declared before the loop
  – Changed within the loop
  – Used after the loop

REVIEW: Another kind of loop
In the book, you read about another, simpler kind of loop

while (CONTINUATION_CONDITION) {
    STATEMENT1;
    STATEMENT2;
    // ...
    STATEMENTn;
}

• Steps Java takes:
  – Check CONTINUATION_CONDITION, maybe skip running the loop body at all
  – Run STATEMENT1 through STATEMENTn
  – Check CONTINUATION_CONDITION, maybe stop running the loop
  – Run STATEMENT1 through STATEMENTn
  – Check CONTINUATION_CONDITION, maybe stop running the loop
  – Run STATEMENT1 through STATEMENTn
  – …and so on until the CONTINUATION_CONDITION is falsified

• If there are variables to set up, or changes to make, we must implement them as separate statements before the loop or in the loop body
Exercise 7.1. Write a class ChangeMaker with a static method getChange which

- Takes a single int argument representing a number of cents
- Returns a string describing the way to represent that amount in the fewest number of common US coins. The string should be of the form "XX quarters, YY dimes, ZZ nickels, WW pennies".

As the usual first step, write a main method with examples and expected results.

How to choose — while or for?

- for loops are great when the loop ranges over one variable, and there is one change to that variable only between passes of the loop
- But otherwise it may be more natural to use a while loop
  - If there are multiple loop variables
  - If there are many changes from pass to pass

Exercise 7.2. TODO — self-describing letters ('a' at position 1, 'b' at 2, etc.)

8 Lecture 8