ICS 111
Nested Loops, Java Methods

- Nested Loops
- Simulations
- Java Methods

ICS 111
Two-Dimensional Problems

- Many problems are best represented using multiple dimensions
- A simple example is a table, in which rows go left to right and columns run top down
- Spreadsheets are similar

ICS 111
Multiplication Table

- With a multiplication table, the product of two numbers a and b is found at the intersection of row a and column b (or viceversa)
- Generally these multiplication tables show the products of all numbers between 1 and 10, or between 1 and 12

ICS 111
Multiplication Table

1:	1	2	3	4	5	6	7	8	9	10
2:	2	4	6	8	10	12	14	16	18	20
3:	3	6	9	12	15	18	21	24	27	30
4:	4	8	12	16	20	24	28	32	36	40
5:	5	10	15	20	25	30	35	40	45	50
6:	6	12	18	24	30	36	42	48	54	60
7:	7	14	21	28	35	42	49	56	63	70
8:	8	16	24	32	40	48	56	64	72	80
9:	9	18	27	36	45	54	63	72	81	90
10:	10	20	30	40	50	60	70	80	90	100

ICS 111
Generating a Multiplication Table

- An outer loop prints each row
- An inner loop prints each value
- both are counting loops that go from 1 to 10 (or 1 to 12)
- each for loop has its own variable: `row`, `col`
- the variable `col`, declared in the inner loop, is only accessible in the body of that inner loop

ICS 111
Multiplication Table: Nested Loops

```
for (int row = 1; row <= 10; row++) {
  System.out.printf ("%2d:", row);
  for (int col = 1; col <= 10; col++) {
    System.out.printf ("%4d", row * col);
  }
  System.out.println();
}
```

- The first printf prints the row header, and could be omitted
- The second printf, in the inner loop, prints the product.
  - the largest product is 100, and
  - %4d specifies 4 characters for each product (d specifies a decimal number), so
  - the first character will always be a space
- The final println ends the row.

ICS 111
Programs that Draw

- A window, or a screen, is a two-dimensional area filled with **pic**ture **el**ements, called **pixel**s
- Filling an area in such a window often requires nested loops

ICS 111
Printing a Calendar

```
Su Mo Tu We Th Fr Sa
          1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

- Easiest to have an outer loop print the weeks, and an inner loop print the days of the week
- Printing all the months in a year might have three nested loops!!!!

ICS 111
Printing a Calendar

```
int weekdayOfFirst = ...; // 0, 1, 2, 3, 4, 5, or 6
int daysInMonth = ...;    // 28, 29, 30, or 31
for (int blank = 0; blank < weekdayOfFirst; blank++) {
  System.out.print ("   ");  // blanks for the days of last month
}
int date = 1;
for (int weekday = weekdayOfFirst; weekday < 7; weekday++) {
  System.out.printf ("%3d", date++);
}
System.out.println ();
while (date <= daysInMonth) {
  for (int weekday = 0; weekday < 7 && date <= daysInMonth; weekday++) {
    System.out.printf ("%3d", date++);
  }
  System.out.println ();
}
```

ICS 111
Printing a Calendar: Alternative

```
int weekdayOfFirst = ...; // 0, 1, 2, 3, 4, 5, or 6
int daysInMonth = ...;    // 28, 29, 30, or 31
int date = 1 - weekdayOfFirst;
while (date <= daysInMonth) {
  for (int weekday = 0; weekday < 7 && date <= daysInMonth; weekday++) {
    if (date >= 1) {
      System.out.printf ("%3d", date++);
    } else {
      System.out.printf ("   ");
      date++;   // don't forget to increment date!
    }
  }
  System.out.println ();
}
```

ICS 111
Simulations

- The world is complicated
- When we use a computer to simulate the real world, we have to simplify
- Instead of having real inputs, we can choose inputs at random in such a way that the random inputs statistically resemble real inputs

ICS 111
Random Numbers

- Math.random() gives a double uniformly distributed between 0 (included) and 1 (excluded)

```
double r = Math.random(); // 0 <= r < 1
```

- if I want a number between 1 and 10, I just multiply and add to give the right range:

```
double oneToTen = Math.random() * 9 + 1;
```

- these numbers are not truly random
- fair dice and fair coin tosses are random
- pseudo-random numbers are the result of a complicated calculation whose results are hard to predict
  - unless you have all the inputs to that calculation

ICS 111
Simulating a Large Shop

- A manager measures how long customers have to wait at the checkout
- The manager wonders how this would change with one more or one fewer cashier
- The average number of customers per day is known
- A program can simulate customers arriving at random times
- The range of times is chosen so the average matches the measured number of customers per day
- The simulation can then measure the wait time with different numbers of cashiers

ICS 111
Java Methods

```
public class Hello {
  public static void main (String[] a) {
    System.out.println ("hello world");
  }
}
```

- `main` is a **method** in java
- Program execution starts with `main`

ICS 111
Familiar Java Methods

We have seen many methods, particularly from the Math library: `Math.round()`, `Math.pow()`, `Math.sqrt()`

- We may call (or **invoke**) these methods because we want the results
  - we want a value that the method computes
- These methods work like mathematical functions: the inputs to the method determine the result
  - method inputs are known as **parameters** or **arguments**
- Or we may call a method because we want it to do something, i.e. have **side effects**: `System.out.println()`
- Some methods both have side effects and also return a result

ICS 111
Calling Java Methods

- When we call a method, we provide the parameters
- After the method completes, the original code resumes execution
  - we say that the method **returns**
  - with or without a return value!
- We will now learn how to write methods
  - again, this is familiar: think of the main method

ICS 111
Writing Java Methods: Overview

- We often don't care how the code does what it does: we treat the method as a **black box**
- Of course, someone had to write the code!
- When creating a method, we have to consider what arguments it takes, and what result (if any) it returns
- We must choose a good name for the method
  - the name should express what the method does
  - in Java, method names use camelCase
- Well-designed methods help in writing well-structured programs

ICS 111
Java Methods: Syntax

```
public static returnType methodName (arguments) {
  body of the method
}
```

- the return type can be `void` if the method doesn't return a value
- arguments are a comma-separated list of *argumentType argumentName*
- for example, the code for Math.pow begins with:

```
public static double pow (double base, double exponent) {
```

- some methods do not have `public static`
  - these will be discussed when we start talking about Objects

ICS 111
A complete Method

```
public static boolean isZero(long value) {
   return (value == 0);
}
```

- This method returns `true` if its parameter is 0, and `false` otherwise

ICS 111
return

```
public static long max(long a, long b) {
  if (a < b) {
    return b;
  }
  return a;
}
```

- When `return` executes, it immediately ends execution of this method, and returns to the caller
  - somewhat like `break` ends execution of a loop
- A method returning a value is required to have a return statement as its last statement
  - in every executable branch
- All return values must be of the correct type

ICS 111
Void Methods

```
public static void
      printTwice(String s) {
  System.out.print(s);
  System.out.println(s);
}
```

- A void method isn't required to have a return statement

ICS 111
Void Methods and `return`

```
public static void printTwice(String s) {
  if (s.length() == 0) {
  // return to the caller, without returning a value
    return;
  }
  System.out.print(s);
  System.out.println(s);
}
```

- When `return` executes, it immediately ends execution of this method, and returns to the caller
- somewhat like `break` ends execution of a loop or switch

ICS 111
Method Parameters

- A method parameter is almost like a variable
- It is a variable initialized by the caller!
- It has a type and a value
- It is entirely local to the method:
  - changing the value of the parameter does not change its value for the caller!
- We will see exceptions to this when we talk about Objects

ICS 111
Locality of Parameters

- Modifying the value of a parameter does not affect the caller
-
```
static void method m (int parm) {
  parm++;  // parm becomes 4
}

int x = 3;
m(x);
// now, still x == 3
```
- Still, methods that don't modify the values of their arguments are often clearer than methods that do

Summary

- Nested loops are appropriate for tables and other two-dimensional problems
- Simulations are useful for predicting what will happen in the real world
- Java methods are building blocks of Java programs
  - methods may or may not return a value
  - methods may have zero or more arguments (parameters)
- You have used and created methods before, but now we begin to look at them more closely