2.2 Libraries and Clients
Libraries

Library: A module whose methods are primarily intended for use by many other programs

Client: Program that calls a library

API: Contract between client and implementation
Random Numbers

“The generation of random numbers is far too important to leave to chance. Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.”

Jon von Neumann (left), ENIAC (right)
Random Numbers

A Weighted Random Number Generator just produced a new batch of numbers.

Let's use them to build narratives!

All Sports Commentary
Standard Random

Our library to generate pseudo-random numbers

```java
public class StdRandom {
    int uniform(int N) {
        // integer between 0 and N-1
    }
    double uniform(double lo, double hi) {
        // real between lo and hi
    }
    boolean bernoulli(double p) {
        // true with probability p
    }
    double gaussian() {
        // normal, mean 0, standard deviation 1
    }
    double gaussian(double m, double s) {
        // normal, mean m, standard deviation s
    }
    int discrete(double[] a) {
        // i with probability a[i]
    }
    void shuffle(double[] a) {
        // randomly shuffle the array a[]
    }

    int getRandomNumber() {
        return 4; // chosen by fair dice roll.
        // guaranteed to be random.
    }
}
```
public class StdRandom {

    // between a and b
    public static double uniform(double a, double b) {
        return a + Math.random() * (b-a);
    }

    // between 0 and N-1
    public static int uniform(int N) {
        return (int) (Math.random() * N);
    }

    // true with probability p
    public static boolean bernoulli(double p) {
        return Math.random() < p;
    }

    // gaussian with mean = 0, stddev = 1
    public static double gaussian() {
        /* see Exercise 1.2.27 */
    }

    // gaussian with given mean and stddev
    public static double gaussian(double mean, double stddev) {
        return mean + (stddev * gaussian());
    }

    ...
}
Unit Testing

Library classes can include testing routines in main()

```java
public class StdRandom {
    ...

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        for (int i = 0; i < N; i++) {
            StdOut.printf(" %2d ", uniform(100));
            StdOut.printf("%8.5f ", uniform(10.0, 99.0));
            StdOut.printf("%5b ", bernoulli(.5));
            StdOut.printf("%7.5f ", gaussian(9.0, .2));
            StdOut.println();
        }
    }
}
```

% java StdRandom 5
61 21.76541 true 9.30910
57 43.64327 false 9.42369
31 30.86201 true 9.06366
92 39.59314 true 9.00896
36 28.27256 false 8.66800
public class RandomPoints {
    public static void main(String args[]) {
        int N = Integer.parseInt(args[0]);
        for (int i = 0; i < N; i++) {
            double x = StdRandom.gaussian(0.5, 0.2);
            double y = StdRandom.gaussian(0.5, 0.2);
            StdDraw.point(x, y);
        }
    }
}

javac RandomPoints.java
java RandomPoints 10000

Using a Library
Statistics
Example Library: Standard Statistics

Library to compute statistics on an array of real numbers

```java
public class StdStats {
    double max(double[] a) { /* largest value */
    double min(double[] a) { /* smallest value */
    double mean(double[] a) { /* average */
    double var(double[] a) { /* sample variance */
    double stddev(double[] a) { /* sample standard deviation */
    double median(double[] a) { /* median */
    void plotPoints(double[] a) { /* plot points at (i, a[i]) */
    void plotLines(double[] a) { /* plot lines connecting points at (i, a[i]) */
    void plotBars(double[] a) { /* plot bars to points at (i, a[i]) */

    \[
    \mu = \frac{a_0 + a_1 + \cdots + a_{n-1}}{n}, \quad \sigma^2 = \frac{(a_0 - \mu)^2 + (a_1 - \mu)^2 + \cdots + (a_{n-1} - \mu)^2}{n - 1}
    \]
    \text{mean} \quad \text{sample variance}
```
public class StdStats {

    public static double max(double[] a) {
        double max = Double.NEGATIVE_INFINITY;
        for (int i = 0; i < a.length; i++)
            if (a[i] > max) max = a[i];
        return max;
    }

    public static double mean(double[] a) {
        double sum = 0.0;
        for (int i = 0; i < a.length; i++)
            sum = sum + a[i];
        return sum / a.length;
    }

    public static double stddev(double[] a) {
        // see text
    }
}
Modular Programming
Modular Programming

Basic principle:
- Divide program into self-contained pieces
- Test each piece individually
- Combine pieces to make program

Example: Flip N coins. How many heads?
- Flip N fair coins and count number of heads
- Repeat simulation, counting number of times each outcome occurs
- Plot histogram of empirical results
- Compare with theoretical predictions

% java Bernoulli 20 100000
public class Bernoulli {

    public static int binomial(int N) {
        int heads = 0;
        for (int j = 0; j < N; j++)
            if (StdRandom.bernoulli(0.5)) heads++;
        return heads;
    }

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int T = Integer.parseInt(args[1]);

        int[] freq = new int[N+1];
        for (int i = 0; i < T; i++)
            freq[binomial(N)]++;

        double[] normalized = new double[N+1];
        for (int i = 0; i <= N; i++)
            normalized[i] = (double) freq[i] / T;
        StdStats.plotBars(normalized);

        double mean = N / 2.0, stddev = Math.sqrt(N) / 2.0;
        double[] phi = new double[N+1];
        for (int i = 0; i <= N; i++)
            phi[i] = Gaussian.phi(i, mean, stddev);
        StdStats.plotLines(phi);
    }
}

flip $N$ fair coins; return # heads
perform $T$ trials of $N$ coin flips each
plot histogram of number of heads
theoretical prediction
Dependency Graph

Modular programming: Build relatively complicated program by combining small, independent modules
Libraries

Why use libraries?

- Makes code easier to understand
- Makes code easier to debug
- Makes code easier to maintain and improve
- Makes code easier to reuse
Extra Slides
Discrete Distribution

Discrete distribution. Given an array of weights (that sum to 1), choose an index at random with probability equal to its weight.

```
public static int discrete(double[] p) {
    // check that weights are nonnegative and sum to 1
    double r = Math.random();
    double sum = 0.0;
    for (int i = 0; i < p.length; i++) {
        sum = sum + p[i];
        if (sum >= r) return i;
    }
    return -1;
}
```

something went wrong