2.2 Libraries and Clients

**Libraries**
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.

**Implementation**
Program that implements the methods in an API.

---

**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.

---

**Standard Random**
Our library to generate pseudo-random numbers.

```java
public class StdRandom {
    // between 0 and 9-1
    public static int uniform(int N) { return Math.random() * N; }

    // between 0 and N-1
    public static int uniform(int N) { return 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() { return gaussian(0.0, 1.0); }

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

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

Include main() to test each library.

```
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

Using a Library

Include main() to test each library.

```
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

Statistics

Ex. Library to compute statistics on an array of real numbers.

```
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 += a[i];
        return sum / a.length;
    }

    public static double stddev(double[] a) {
        double sum = 0.0;
        for (int i = 0; i < a.length; i++)
            sum += a[i];
        double mean = mean(a);
        double sumsq = 0.0;
        for (int i = 0; i < a.length; i++)
            sumsq += (a[i] - mean) * (a[i] - mean);
        return Math.sqrt(sumsq / (a.length - 1));
    }
}
```

Standard Statistics

Ex. Library to compute statistics on an array of real numbers.

```
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 += a[i];
        return sum / a.length;
    }

    public static double stddev(double[] a) {
        double sum = 0.0;
        for (int i = 0; i < a.length; i++)
            sum += a[i];
        double mean = mean(a);
        double sumsq = 0.0;
        for (int i = 0; i < a.length; i++)
            sumsq += (a[i] - mean) * (a[i] - mean);
        return Math.sqrt(sumsq / (a.length - 1));
    }
}
```

Modular Programming
Modular Programming

Modular programming
- Divide program into self-contained pieces.
- Test each piece individually.
- Combine pieces to make program.

Ex. Flip N coins. How many heads?
- Read arguments from user.
- Flip one fair coin.
- 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.

Bernoulli Trials

```java
public class Bernoulli {
    //...
    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);
    }
}
```

Dependencies Graph

Modular programming. Build relatively complicated program by combining several small, independent, modules.

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.