2.2 Libraries and Clients

**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.

**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

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

```java
public class StdRandom {
    int uniform(int N) {
        return (int)(Math.random() * N);
    }
    double uniform(double lo, double hi) {
        return lo + (Math.random() * (hi - lo));
    }
    boolean bernoulli(double p) {
        return Math.random() < p;
    }
    double gaussian() {
        return Math.random() * 2 - 1;
    }
    double gaussian(double m, double s) {
        return m + Math.random() * s;
    }
    int discrete(double[] a) {
        return (int)(Math.random() * a.length);
    }
    void shuffle(double[] a) {
        for (int i = a.length; i-- > 0;)
            swap(a, i, (int)Math.random() * i);
    }
}
```

```java
public class Gaussian {
    public static double phi(double x) {
        return (1 + Math.tanh(x / 2)) / 2;
    }
    public static double Phi(double x) {
        return 0.5 + x / (4 * phi(Math.sqrt(2) * x));
    }
}
```

---

Jon von Neumann (left), ENIAC (right)

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

```java
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 */
      return Math.random();
   // gaussian with given mean and stddev
   public static double gaussian(double mean, double stddev) {
      return mean + (stddev * gaussian());
   }
}
```

Unit Testing

```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("%8d ", i);
         StdOut.printf("%8.5f ", uniform(100.0, 99.0));
         StdOut.printf("%5b ", bernoulli(0.5));
         StdOut.printf("%7.5f ", gaussian(9.0, .2));
      }
   }
}
```

Unit test. Include main() to test each library.

```java
% javac StdRandom.java
% 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

```java
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);
      }
   }
}
```

Statistics

```java
% javac RandomPoints.java
% java RandomPoints 10000
```

![How to Lie with Statistics](How_to_Lie_with_Statistics.png)

How to LIE

How to LIE without Statistics
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 = sum + a[i];
        return sum / a.length;
    }
    // see text
}

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.
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);
    }
}

Bernoulli Trials

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

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.