1.3 Conditionals and Loops
A Foundation for Programming

Any program you might want to write

- objects
- functions and modules
- graphics, sound, and image I/O
- arrays
- conditionals and loops

- Math
- text I/O

- primitive data types
- assignment statements

Equivalent to a calculator
A Foundation for Programming

any program you might want to write

objects

functions and modules

graphics, sound, and image I/O

arrays

conditionals and loops

Math
text I/O

primitive data types

assignment statements

to infinity and beyond!
Control Flow

**Control flow.**
- Sequence of statements that are actually executed in a program.
- Conditionals and loops: enable us to choreograph control flow.

straight-line control flow

control flow with conditionals and loops
Conditionals
The if statement. A common branching structure.

- Check boolean condition.
- If true, execute some statements.
- If false, execute other statements.
The **if** statement. A common branching structure.

- **Check** boolean condition.
- **If** true, execute some statements.
- **If** false, execute other statements.

![Diagram of if statement with boolean expression and sequence of statements.](image-url)
If Statement

Ex. Take different action depending on value of variable.

```java
public class Flip {
    public static void main(String[] args) {
        if (Math.random() < 0.5) System.out.println("Heads");
        else System.out.println("Tails");
    }
}
```
## If Statement Examples

<table>
<thead>
<tr>
<th>absolute value</th>
<th>if ( (x &lt; 0) )  ( x = -x; )</th>
</tr>
</thead>
</table>
| put \( x \) and \( y \) into sorted order | if \( (x > y) \)  
{
    int \( t = x; \)
    \( y = x; \)
    \( x = t; \)
} |
| maximum of \( x \) and \( y \) | if \( (x > y) \)  \( \text{max} = x; \)  
else \( \text{max} = y; \) |
| error check for division operation | if \( (\text{den} == 0) \)  
System.out.println("Division by zero");  
else  
System.out.println("Quotient = " + \( \text{num}/\text{den} \)); |
| error check for quadratic formula | double discriminant = \( b^2 - 4.0 \times c; \)  
if \( (\text{discriminant} < 0.0) \)  
{
    System.out.println("No real roots");
}  
else  
{
    System.out.println((-\( b + \text{Math.sqrt(discriminant)} \))/2.0);  
    System.out.println((-\( b - \text{Math.sqrt(discriminant)} \))/2.0);  
} |
The While Loop
The while loop. A common repetition structure.
- Check a boolean expression.
- Execute a sequence of statements.
- Repeat.

```java
while (boolean expression) {
    statement 1;
    statement 2;
}
```
While Loops: Powers of Two

Ex. Print first \( n \) powers of 2.

- Increment \( i \) from 1 to \( n \).
- Double \( v \) each time.

```java
int i = 0;
int v = 1;
while (i <= N) {
    System.out.println(v);
    i = i + 1;
    v = 2 * v;
}
```

<table>
<thead>
<tr>
<th>( i )</th>
<th>( v )</th>
<th>( i &lt;= N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>true</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>true</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>true</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>true</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>true</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>true</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>false</td>
</tr>
</tbody>
</table>

\( n = 6 \)
public class PowersOfTwo {
    public static void main(String[] args) {

        // last power of two to print
        int N = Integer.parseInt(args[0]);

        int i = 0;  // loop control counter
        int v = 1;  // current power of two
        while (i <= N) {
            System.out.println(v);
            i = i + 1;
            v = 2 * v;
        }
    }
}
Q. Anything wrong with the following code for printing powers of 2?

```java
int i = 0;
int v = 1;
while (i <= N)
    System.out.println(v);
    i = i + 1;
    v = 2 * v;
```
Q. Anything wrong with the following code for printing powers of 2?

```java
int i = 0;
int v = 1;
while (i <= N)
    System.out.println(v);
    i = i + 1;
    v = 2 * v;
```

A. Need curly braces around statements in while loop; otherwise it enters an infinite loop, printing 1s.

Moment of panic. How to stop infinite loop?
"A wonderful square root. Let’s hope it can be used for the good of mankind."

Copyright 2004, Sidney Harris, http://www.sciencecartoonsplus.com
While Loops: Square Root

Q. How might we implement Math.sqrt()?
A. To compute the square root of \( c \):
- Initialize \( t_0 = c \).
- Repeat until \( t_i = \frac{c}{t_i} \), up to desired precision:
  set \( t_{i+1} \) to be the average of \( t_i \) and \( \frac{c}{t_i} \).

\[
\begin{align*}
  t_0 & = 2.0 \\
  t_1 & = \frac{1}{2}(t_0 + \frac{2}{t_0}) = 1.5 \\
  t_2 & = \frac{1}{2}(t_1 + \frac{2}{t_1}) = 1.416666666666665 \\
  t_3 & = \frac{1}{2}(t_2 + \frac{2}{t_2}) = 1.4142156862745097 \\
  t_4 & = \frac{1}{2}(t_3 + \frac{2}{t_3}) = 1.4142135623746899 \\
  t_5 & = \frac{1}{2}(t_4 + \frac{2}{t_4}) = 1.414213562373095
\end{align*}
\]

computing the square root of 2
### While Loops: Square Root

**Q.** How might we implement `Math.sqrt()`?

**A.** To compute the square root of `c`:

- Initialize $t_0 = c$.
- **Repeat until** $t_i = c / t_i$, **up to desired precision**:
  - Set $t_{i+1}$ to be the average of $t_i$ and $c / t_i$.

```java
public class Sqrt {
    public static void main(String[] args) {
        double EPS = 1E-15;
        double c = Double.parseDouble(args[0]);
        double t = c;
        while (Math.abs(t - c/t) > t*EPS) {
            t = (c/t + t) / 2.0;
        } System.out.println(t);
    }
}
```

% java Sqrt 2.0
1.414213562373095

15 decimal digits of accuracy in 5 iterations
Newton-Raphson Method

Square root method explained.

- Goal: find root of function \( f(x) \).
- Start with estimate \( t_0 \).
- Draw line tangent to curve at \( x = t_i \).
- Set \( t_{i+1} \) to be \( x \)-coordinate where line hits \( x \)-axis.
- Repeat until desired precision.

\[ f(x) = x^2 - c \] to compute \( \sqrt{c} \)
The For Loop

```c
#include <stdio.h>
int main(void)
{
  int count;
  for (count = 1; count <= 500; count++)
    printf("I will not throw paper airplanes in class.");
  return 0;
}
```
For Loops

The **for** loop. Another common repetition structure.
- Execute initialization statement.
- Check boolean expression.
- Execute sequence of statements.
- Execute increment statement.
- Repeat.

```plaintext
for (init; boolean expression; increment) {
    statement 1;
    statement 2;
}
```
Anatomy of a For Loop

Q. What does it print?
A.
Create subdivision of a ruler.

- Initialize `ruler` to empty string.
- For each value `i` from 1 to `N`:
  sandwich two copies of `ruler` on either side of `i`.

```java
public class Ruler {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        String ruler = " ";
        for (int i = 1; i <= N; i++) {
            ruler = ruler + i + ruler;
        }
        System.out.println(ruler);
    }
}
```

<table>
<thead>
<tr>
<th>i</th>
<th>ruler</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot; 1 &quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot; 1 2 1 &quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot; 1 2 1 3 1 2 1 &quot;</td>
</tr>
</tbody>
</table>
For Loops: Subdivisions of a Ruler

Observation. Loops can produce a huge amount of output!
## Loop Examples

<table>
<thead>
<tr>
<th>Loop Example</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>print powers of two</strong></td>
<td><code>int v = 1;</code>&lt;br&gt;for (int i = 0; i &lt;= N; i++)&lt;br&gt;  <code>{&lt;br&gt;    System.out.println(i + &quot; &quot; + v);&lt;br&gt;    v = 2*v;&lt;br&gt;  }</code></td>
</tr>
<tr>
<td><strong>print largest power of two less than or equal to N</strong></td>
<td><code>int v = 1;</code>&lt;br&gt;while (v &lt;= N/2)&lt;br&gt;  <code>{&lt;br&gt;    v = 2*v;&lt;br&gt;    System.out.println(v);&lt;br&gt;  }</code></td>
</tr>
<tr>
<td><strong>compute a finite sum ((1 + 2 + \ldots + N))</strong></td>
<td><code>int sum = 0;</code>&lt;br&gt;for (int i = 1; i &lt;= N; i++)&lt;br&gt;  <code>{&lt;br&gt;    sum += i;&lt;br&gt;    System.out.println(sum);&lt;br&gt;  }</code></td>
</tr>
<tr>
<td><strong>compute a finite product ((N! = 1 \times 2 \times \ldots \times N))</strong></td>
<td><code>int product = 1;</code>&lt;br&gt;for (int i = 1; i &lt;= N; i++)&lt;br&gt;  <code>{&lt;br&gt;    product *= i;&lt;br&gt;    System.out.println(product);&lt;br&gt;  }</code></td>
</tr>
<tr>
<td><strong>print a table of function values</strong></td>
<td><code>for (int i = 0; i &lt;= N; i++)&lt;br&gt;  System.out.println(i + &quot; &quot; + 2*Math.PI*i/N);</code></td>
</tr>
<tr>
<td><strong>print the ruler function (see Program 1.2.1)</strong></td>
<td><code>String ruler = &quot; &quot;;&lt;br&gt;for (int i = 1; i &lt;= N; i++)&lt;br&gt;  </code>{&lt;br&gt;    ruler = ruler + i + ruler;&lt;br&gt;    System.out.println(ruler);&lt;br&gt;  }`</td>
</tr>
</tbody>
</table>
Nesting
Nesting Conditionals and Loops

**Conditionals** enable you to do one of $2^n$ sequences of operations with $n$ lines.

```java
if (a0 > 0) System.out.print(0);
if (a1 > 0) System.out.print(1);
if (a2 > 0) System.out.print(2);
if (a3 > 0) System.out.print(3);
if (a4 > 0) System.out.print(4);
if (a5 > 0) System.out.print(5);
if (a6 > 0) System.out.print(6);
if (a7 > 0) System.out.print(7);
if (a8 > 0) System.out.print(8);
if (a9 > 0) System.out.print(9);
```

**Loops** enable you to do an operation $n$ times using only 2 lines of code.

```java
double sum = 0.0;
for (int i = 1; i <= 1024; i++)
    sum = sum + 1.0 / i;
```

This computes $1/1 + 1/2 + \ldots + 1/1024$

$2^{10} = 1024$ possible results, depending on input

More sophisticated programs.
- Nest conditionals within conditionals.
- Nest loops within loops.
- Nest conditionals within loops within loops.
Nested If Statements

**Ex.** Pay a certain tax rate depending on income level.

<table>
<thead>
<tr>
<th>Income</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 47,450</td>
<td>22%</td>
</tr>
<tr>
<td>47,450 - 114,650</td>
<td>25%</td>
</tr>
<tr>
<td>114,650 - 174,700</td>
<td>28%</td>
</tr>
<tr>
<td>174,700 - 311,950</td>
<td>33%</td>
</tr>
<tr>
<td>311,950 -</td>
<td>35%</td>
</tr>
</tbody>
</table>

5 mutually exclusive alternatives

double rate;
if  (income < 47450)  rate = 0.22;
else if (income < 114650)  rate = 0.25;
else if (income < 174700)  rate = 0.28;
else if (income < 311950)  rate = 0.33;
else  rate = 0.35;

graduated income tax calculation
Nested If Statements

```c
if      (income <  47450) rate = 0.22;
else if (income < 114650) rate = 0.25;
else if (income < 174700) rate = 0.28;
else if (income < 311950) rate = 0.33;
else if (income < 311950) rate = 0.35;
```

is shorthand for

```c
if (income <  47450) rate = 0.22;
else {
    if (income < 114650) rate = 0.25;
    else {
        if (income < 174700) rate = 0.28;
        else {
            if (income < 311950) rate = 0.33;
            else if (income < 311950) rate = 0.35;
        }
    }
}
```

Be careful when nesting if-else statements (see Q+A p. 75).
Q. Anything wrong with the following for income tax calculation?

```java
double rate = 0.35;
if (income < 47450) rate = 0.22;
if (income < 114650) rate = 0.25;
if (income < 174700) rate = 0.28;
if (income < 311950) rate = 0.33;
```

Wrong graduated income tax calculation
Monte Carlo Simulation
Gambler's Ruin

Gambler's ruin. Gambler starts with \$stake and places \$1 fair bets until going broke or reaching \$goal.

- What are the chances of winning?
- How many bets will it take?

One approach. Monte Carlo simulation.

- Flip digital coins and see what happens.
- Repeat and compute statistics.
public class Gambler {
    public static void main(String[] args) {
        int stake = Integer.parseInt(args[0]);
        int goal  = Integer.parseInt(args[1]);
        int trials = Integer.parseInt(args[2]);
        int wins  = 0;

        // repeat experiment N times
        for (int i = 0; i < trials; i++) {
            // do one gambler's ruin experiment
            int t = stake;
            while (t > 0 && t < goal) {
                // flip coin and update
                if (Math.random() < 0.5) t++;
                else t--;
            }
            if (t == goal) wins++;
        }
        System.out.println(wins + " wins of " + trials);
    }
}
Digression: Simulation and Analysis

Fact. Probability of winning = stake ÷ goal.
Fact. Expected number of bets = stake × desired gain.
Ex. 20% chance of turning $500 into $2500, but expect to make one million $1 bets.

Remark. Both facts can be proved mathematically; for more complex scenarios, computer simulation is often the best plan of attack.
Control Flow Summary

Control flow.
- Sequence of statements that are actually executed in a program.
- Conditionals and loops: enables us to choreograph the control flow.

<table>
<thead>
<tr>
<th>Control Flow</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Straight-line programs</td>
<td>All statements are executed in the order given.</td>
<td></td>
</tr>
<tr>
<td>Conditionals</td>
<td>Certain statements are executed depending on the values of certain variables.</td>
<td>if, if-else</td>
</tr>
<tr>
<td>Loops</td>
<td>Certain statements are executed repeatedly until certain conditions are met.</td>
<td>while, for, do-while</td>
</tr>
</tbody>
</table>
Program Development

Ada Lovelace

Admiral Grace Murray Hopper
Program development. Creating a program and putting it to good use.
Def. A bug is a mistake in a computer program.

Programming is primarily a process of finding and fixing bugs.

Good news. Can use computer to test program.
Bad news. Cannot use computer to automatically find all bugs.
95% of Program Development

Debugging. Cyclic process of editing, compiling, and fixing errors.
- Always a logical explanation.
- What would the machine do?
- Explain it to the teddy bear.

You will make many mistakes as you write programs. It's normal.

“As soon as we started programming, we found out to our surprise that it wasn't as easy to get programs right as we had thought. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.” — Maurice Wilkes

“If I had eight hours to chop down a tree, I would spend six hours sharpening an axe.” — Abraham Lincoln
Debugging Example

**Factor.** Given an integer $N > 1$, compute its prime factorization.

\[
3,757,208 = 2^3 \times 7 \times 13^2 \times 397
\]

\[
98 = 2 \times 7^2
\]

\[
17 = 17
\]

\[
11,111,111,111,111,111 = 2,071,723 \times 5,363,222,357
\]

**Application.** Break RSA cryptosystem (factor 200-digit numbers).
Factor. Given an integer \( N \), compute its prime factorization.

Brute-force algorithm. For each putative factor \( i = 2, 3, 4, \ldots \), check if \( N \) is a multiple of \( i \), and if so, divide it out.
Debugging: 95% of Program Development

Programming. A process of finding and fixing mistakes.
- Compiler error messages help locate syntax errors.
- Run program to find semantic and performance errors.

```java
public class Factors {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (int i = 0; i < N; i++) {
            while (N % i == 0) {
                System.out.print(i + " ");
                N = N / i;
            }
        }
    }
}
```

this program has many bugs!
Debugging: Syntax Errors

Syntax error. Illegal Java program.
- Compiler error messages help locate problem.
- Goal: no errors and a file named Factors.class.
### Debugging: Syntax Errors

**Syntax error.** Illegal Java program.
- Compiler error messages help locate problem.
- **Goal:** no errors and a file named `Factors.class`.

```java
public class Factors {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (int i = 0; i < N; i++) {
            while (N % i == 0)
                System.out.print(i + " ");
            N = N / i;
        }
    }
}
```

**Syntax (compile-time) errors**
- need to declare variable i
- need terminating semicolons
Debugging: Semantic Errors

**Semantic error.** Legal but wrong Java program.
- Run program to identify problem.
- Add print statements if needed to produce trace.

```java
public class Factors {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (int i = 0; i < N; i++) {
            while (N % i == 0)
                System.out.print(i + " ");
            N = N / i;
        }
    }
}
```

```
javac Factors.java
java Factors
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 0
    at Factors.main(Factors.java:5)
```
Debugging: Semantic Errors

**Semantic error.** Legal but wrong Java program.
- Run program to identify problem.
- Add print statements if needed to produce trace.

```java
public class Factors {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (int i = 0; i < N; i++) {
            while (N % i == 0)
                System.out.print(i + " ");
            N = N / i;
        }
    }
}
```

%% javac Factors.java
%% java Factors 98
Exception in thread "main"
java.lang.ArithmeticException: / by zero
    at Factors.main(Factors.java:8)

need to start at 2 because 0 and 1 cannot be factors
Debugging: Semantic Errors

Semantic error. Legal but wrong Java program.
- Run program to identify problem.
- Add print statements if needed to produce trace.

```java
public class Factors {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (int i = 2; i < N; i++) {
            while (N % i == 0) {
                System.out.print(i + " ");
                N = N / i;
            }
        }
    }
}
```

```
javac Factors.java
java Factors 98
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ...
public class Factors {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (int i = 2; i < N; i++) {
            while (N % i == 0) {
                System.out.print(i + " ");
                N = N / i;
            }
        }
    }
}

Success. Program factors $98 = 2 \times 7^2$.
- But that doesn't mean it works for all inputs.
- Add trace to find and fix (minor) problems.

```
% java Factors 98
2 7 %  
% java Factors 5
% java Factors 6
2 %
```
Success. Program factors $98 = 2 \times 7^2$.
- But that doesn't mean it works for all inputs.
- Add trace to find and fix (minor) problems.

```java
public class Factors {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (int i = 2; i < N; i++) {
            while (N % i == 0) {
                System.out.println(i + " ");
                N = N / i;
            }
            System.out.println("TRACE: " + i + " " + N);
        }
    }
}
```

Aha! Add trace to find and fix (minor) problems.

Debugging: The Beat Goes On

% java Factors 5
TRACE 2 5
TRACE 3 5
TRACE 4 5

% java Factors 6
2
TRACE 2 3

Aha! Print out N after for loop if it is not 1.
Success. Program seems to work.

dpublic class Factors {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (int i = 2; i < N; i++) {
            while (N % i == 0) {
                System.out.print(i + " ");
                N = N / i;
            }
        }
        if (N > 1) System.out.println(N);
        else System.out.println();
    }
}

Success. Program seems to work.

% java Factors 5
5
% java Factors 6
2 3
% java Factors 98
2 7 7
% java Factors 3757208
2 2 2 7 13 13 397

"corner case"
Performance error. Correct program, but too slow.

```java
public class Factors {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (int i = 2; i < N; i++) {
            while (N % i == 0) {
                System.out.print(i + " ");
                N = N / i;
            }
        }
        if (N > 1) System.out.println(N);
        else System.out.println();
    }
}
```

Debugging: Performance Error

```
% java Factors 11111111
11 73 11 137

% java Factors 11111111111
21649 51329

% java Factors 11111111111111
11 239 4649 909091

% java Factors 11111111111111111
2071723

very long wait (with a surprise ending)
```
Debugging: Performance Error

Performance error. Correct program, but too slow.

Solution. Improve or change underlying algorithm.

```java
public class Factors {
    public static void main(String[] args) {
        long N = Long.parseLong(args[0]);
        for (int i = 2; i <= N/i; i++) {
            while (N % i == 0) {
                System.out.print(i + " ");
                N = N / i;
            }
        }
        if (N > 1) System.out.println(N);
        else System.out.println();
    }
}
```

% java Factors 11111111
11 73 11 137

% java Factors 111111111111
21649 51329

% java Factors 11111111111111111
11 239 4649 909091

% java Factors 1111111111111111111111
2071723 5363222357

fixes performance error: if N has a factor, it has one less than or equal to its square root
**Program Development: Analysis**

**Q.** How large an integer can I factor?

<table>
<thead>
<tr>
<th>digits</th>
<th>( i \leq N )</th>
<th>( (i \times i) \leq N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>instant</td>
<td>instant</td>
</tr>
<tr>
<td>6</td>
<td>0.15 seconds</td>
<td>instant</td>
</tr>
<tr>
<td>9</td>
<td>77 seconds</td>
<td>instant</td>
</tr>
<tr>
<td>12</td>
<td>21 hours ( \dagger )</td>
<td>0.16 seconds</td>
</tr>
<tr>
<td>15</td>
<td>2.4 years ( \dagger )</td>
<td>2.7 seconds</td>
</tr>
<tr>
<td>18</td>
<td>2.4 millennia ( \dagger )</td>
<td>92 seconds</td>
</tr>
</tbody>
</table>

- \( \dagger \) estimated

---

**Note.** Can't break RSA this way (experts are still trying).
Debugging

Programming. A process of finding and fixing mistakes.

1. Create the program.

2. Compile it.
   Compiler says: That’s not a legal program.
   Back to step 1 to fix syntax errors.

3. Execute it.
   Result is bizarrely (or subtly) wrong.
   Back to step 1 to fix semantic errors.

4. Enjoy the satisfaction of a working program!

5. Too slow? Back to step 1 to try a different algorithm.
U.S.S. Grace Murray Hopper
Extra Slides
Oblivious Sorting

**Sort.** Read in 3 integers and rearrange them in ascending order.

```java
public class Sort3 {
    public static void main(String[] args) {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        int c = Integer.parseInt(args[2]);

        if (b > c) { int t = b; b = c; c = t; }
        if (a > b) { int t = a; a = b; b = t; }
        if (b > c) { int t = b; b = c; c = t; }

        System.out.println(a + " " + b + " " + c);
    }
}
```

Puzzle 1. Sort 4 integers with 5 compare-exchanges.
Puzzle 2. Sort 6 integers with 12.
The **do-while loop**. A less common repetition structure.

- Execute sequence of statements.
- Check loop-continuation condition.
- Repeat.

```java
do {
    statement 1;
    statement 2;
} while (boolean expression);
```

**do-while loop syntax**
Do-While Loop

**Ex.** Find a point \((x, y)\) that is uniformly distributed in unit disc.

- Pick a random point in unit square.
- Check if point is also in unit disc.
- Repeat.

```java
do {
    x = 2.0 * Math.random() - 1.0;
    y = 2.0 * Math.random() - 1.0;
} while (x*x + y*y > 1.0);
```

between -1 and 1