Review

• Primitive Data Types & Variables
  – int, long
  – float, double
  – boolean
  – char

• String

• Mathematical operators: +  -  *  /  %

• Comparison: <  >  <=  >=  ==
1.3 Conditionals and Loops
Control Flow

- Programs execute one statement after another
- Conditionals and loops allow us to control the flow

straight-line control flow

control flow with conditionals and loops
Conditionals
If Statement

• The **if** statement: A common branching structure
  – Evaluate a **boolean** expression
  – If **true**, execute some statements
  – If **false**, execute other statements

if (boolean expression) {
  //statement T;
}
else {
  //statement F;
}

can be any sequence of statements
A program to check if a number is even or odd?

• How do we provide a number to java?
• Command-line arguments
  – Remembers that args[0] is the first argument, args[1] the second argument and so on
  – Also remember that args[0] is a String
Command line arguments

• To run programs you have written so far
  – java MyHouse
  – java HelloWorld
• We’d like to be able to provide information at the command line
  – java HelloWorld Arvind
• want the program say Hello Arvind as opposed Hello World.
public class Hello {
    public static void main (String[] args) {
        String name = args[0];
        System.out.println("Hello there "+ name);
    }
}
Command line arguments

• args[0] will be a String
• How to convert a String to an integer?
• Integer.parseInt
Back to even or odd detection

Live coding ....
Relational Expressions

< less than
> is greater than
<= is less than or equal to
>= is greater than or equal to
== is equivalent
!= is not equivalent
Relational Expressions: Examples

1. if ( true ) { ... }
2. if ( 10 > 10 ) { ... }
3. if ( 10 >= 10 ) { ... }
4. if ( 'a' == 'a' ) { ... }
5. if ( 'a' != 'a' ) { ... }
6. if ( "Penn" != "penn") { ... }
Logical Expressions

&& logical conjunction (and)
• both expressions must be true for conjunction to be true

|| logical disjunction (or)
• either expression must be true for disjunction to be true

! logical negation (not)
• true $\rightarrow$ false, false $\rightarrow$ true
Logical Expression Examples

1. if ((2 > 1) && (3 > 4)) { ... }
2. if ((‘b’ == ‘b’) && (1 + 2 == 3)) { ... }
3. if (!false) { ... }
4. if (! (1 < -1)) { ... }
5. if (! (10 < 20) || false) { ... }
6. if (! (10 > 20) && (10 < 20)) { ... }
7. if (true || false) && true) { ... }
8. if (true && false) || true) { ... }
9. ...
If Statement

• The **if** statement: A common branching structure
  – Evaluate a **boolean** expression
  – If **true**, execute some statements
  – If **false**, execute other statements
If Statement

• Ex. Take different actions depending on the value of a 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");
      }
   }
}
```

% java Flip
Heads
% java Flip
Heads
% java Flip
Tails
% java Flip
Heads
## If Statement Examples

<table>
<thead>
<tr>
<th>absolute value</th>
<th>if ((x &lt; 0)) (x = -x;)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>put x and y into sorted order</strong></td>
<td>if ((x &gt; y))</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>\hspace{1em} \text{int} t = x;</td>
</tr>
<tr>
<td></td>
<td>\hspace{1em} x = y;</td>
</tr>
<tr>
<td></td>
<td>\hspace{1em} y = t;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td><strong>maximum of x and y</strong></td>
<td>if ((x &gt; y)) (max = x;)</td>
</tr>
<tr>
<td></td>
<td>else (max = y;)</td>
</tr>
<tr>
<td><strong>error check for division operation</strong></td>
<td>if ((\text{den} == 0)) System.out.println(&quot;Division by zero&quot;);</td>
</tr>
<tr>
<td></td>
<td>else System.out.println(&quot;Quotient = &quot; + num/den);</td>
</tr>
<tr>
<td><strong>error check for quadratic formula</strong></td>
<td>double discriminant = b<em>b - 4.0</em>c;</td>
</tr>
<tr>
<td></td>
<td>if ((\text{discriminant} &lt; 0.0))</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>\hspace{1em} System.out.println(&quot;No real roots&quot;);</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>else {</td>
</tr>
<tr>
<td></td>
<td>\hspace{1em} System.out.println((-b + Math.sqrt(discriminant))/2.0);</td>
</tr>
<tr>
<td></td>
<td>\hspace{1em} System.out.println((-b - Math.sqrt(discriminant))/2.0);</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>
if ( boolean_expression_1 ) {
    statements;
} else if ( boolean_expression_2 ) {
    statements;
} else if ( boolean_expression_3 ) {
    statements;
} else {
    statements;
}
Example: Graduated Income Tax

Pay a certain income tax rate depending on income:

<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
Nested If Statements

Use nested if statements to handle multiple alternatives

```java
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;
        }
    }
}
```

<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>
Nested If Statements

5 mutually exclusive alternatives

<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>

Alternative shortened version:

```java
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;
```
Nested If Statements

What is wrong with the following implementation?

```
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;
```

5 mutually exclusive alternatives

<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>
An aside ... Operators

+, -, *, / and ... 

```plaintext
i++;  equivalent to  i = i + 1;
i += 2;  equivalent to  i = i + 2;
i--;  equivalent to  i = i - 1;
i -= 3;  equivalent to  i = i - 3;
i *= 2;  equivalent to  i = i * 2;
i /= 4;  equivalent to  i = i / 4;
i % 3;  remainder after i is divided by 3 (modulo)
```
Iteration
Iteration

Repetition of a program block

• Iterate when a block of code is to be repeated multiple times.

Options

• The for-loop
• The while-loop
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.\n");
    return 0;
}
```
For Loops

• Handles details of the counter-controlled loop “automatically”
• The for loop structure includes:
  – the initialization of the the loop control variable,
  – the termination condition test, and
  – control variable modification

```java
for (int i = 1; i < 101; i = i + 1) {
    // initialization
    test
    // modification
}
```
When Does a *for* Loop Initialize, Test and Modify?

• A for loop
  – initializes the loop control variable before beginning the first loop iteration
  – performs the loop termination test before each iteration of the loop
  – modifies the loop control variable at the **very end** of each iteration of the loop
For Loop: Powers of Two

Example: Print powers of 2 that are $\leq 2^N$
- Increment $i$ from 0 to $N$
- Double $v$ each time

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

Output:
```
  0 1
  1 2
  2 4
  3 8
  4 16
```

$N = 4$
For Loop Examples

• A for loop that counts from 0 to 9:

  // modify part can be simply “i++”
  for ( i = 0;  i < 10;  i = i + 1 ) {
    System.out.println( i ) ;
  }

• …or we can count backwards by 2’s:

  // modify part can be “i -= 2”
  for ( i = 10;  i > 0;  i = i - 2 ) {
    System.out.println( i ) ;
  }
## For loop examples

<table>
<thead>
<tr>
<th>Description</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>compute a finite sum ((1 + 2 + \ldots + N))</td>
<td>int sum = 0;</td>
</tr>
<tr>
<td></td>
<td>for (int i = 0; i &lt;= N; i++)</td>
</tr>
<tr>
<td></td>
<td>sum += i;</td>
</tr>
<tr>
<td></td>
<td>System.out.println(sum);</td>
</tr>
<tr>
<td>print largest power of two less than or equal to (N)</td>
<td>int v = 0;</td>
</tr>
<tr>
<td></td>
<td>for (v = 1; v &lt;= N/2; v *= 2);</td>
</tr>
<tr>
<td></td>
<td>System.out.println(v);</td>
</tr>
</tbody>
</table>
While Loop

The **while** loop: A common repetition structure

- Evaluate a **boolean** expression
- If **true**, execute some statements
- Repeat

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

**loop continuation condition**
What will this do?

System.out.print("Program running");
while (true) {
   System.out.print(".");
}
System.out.println();
System.out.println("Program Exiting");
While Loop: Powers of Two

Example: Print powers of 2 that are $\leq 2^N$
- Increment $i$ from 0 to $N$
- Double $v$ each time

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

Output:

<table>
<thead>
<tr>
<th>$i$</th>
<th>$v$</th>
<th>$i \leq N$</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>false</td>
</tr>
</tbody>
</table>

$N = 4$
While Loop Challenge

Q: Is there anything wrong with the following code for printing powers of 2?

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

Q: Is there anything wrong with the following code for printing powers of 2?

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

A: Need curly braces around statements in while loop
    • otherwise it enters an infinite loop, printing "0 1"
The 3 Parts of a Loop

... 
int i = 1 ;  \textcolor{blue}{\text{initialization of loop control variable}}

// count from 1 to 100 
while ( i < 101 ) {
    \textcolor{blue}{\text{test of loop termination condition}}
    System.out.println( i ) ;
    \textcolor{blue}{\text{modification of loop control variable}}
    i = i + 1 ;
}

// count from 1 to 100
while ( i < 101 ) {
    System.out.println( i ) ;
    i = i + 1 ;
}
Example: Factorial

...  
int factorial = 1;
while (myNumber > 0) {
    factorial *= myNumber;
    --myNumber;
}
System.out.println(factorial);
Keyboard input

• PennDraw.hasNextKeyTyped() – check to see if the user has pressed key

• If the user presses a key, PennDraw.hasNextKeyTyped() is true until and unless you write a line that processes the input

• c = PennDraw.nextKeyTyped();
public class KeyBoardInput {
    public static void main(String[] args) {
        char c = 0;
        double radius = 0.02;
        PennDraw.setCanvasSize(600, 600);
        PennDraw.enableAnimation(10);
        while (c != 'q') {
            if (PennDraw.hasNextKeyTyped()) {
                c = PennDraw.nextKeyTyped();
            }
            PennDraw.circle(0.5, 0.5, radius);
            radius = radius + 0.02;
            PennDraw.advance();
        }
    }
}
Using PennDraw for animation

• PennDraw.enableAnimation(10)
  Animation at 10 frames per second

• PennDraw.advance()

• PennDraw.disableAnimation()
The break & continue Statements

• The break & continue statements can be used in while and for loops to skip the remaining statements in the loop body:
  – break causes the looping itself to abort
  – continue causes the next turn of the loop to start

• In a for loop, the modification step will still be executed
Example: Break in a For-Loop

... 
int i;
for (i = 1; i < 10; i = i + 1) {
   if (i == 5) {
       break;
   }
   System.out.println(i);
}
System.out.println("\nBroke out of loop at i = "+ i);

OUTPUT:

1 2 3 4
Broke out of loop at i = 5
Example: Continue in a For-Loop

... 
int i;
for (i = 1; i < 10; i = i + 1) {
    if (i == 5) {
        continue;
    }
    System.out.println(i);
}
System.out.println("Done");

OUTPUT:
1 2 3 4 6 7 8 9
Done
Problem: Continue in While-Loop

// This seems equivalent to for loop
// in previous slide—but is it??
...
int i = 1;
while (i < 10) {
    if (i == 5) {
        continue;
    }
    System.out.println(i);
    i = i + 1;
}
System.out.println("Done");

/**/
Variable Scope

**Variable scope:**

- That set of code statements in which the variable is known to the compiler
- Where it can be referenced in your program
- Limited to the *code block* in which it is defined
  - A *code block* is a set of code enclosed in braces ({ })

One interesting application of this principle allowed in Java involves the *for loop* construct
Scoping and the For-Loop Index

• Can declare and initialize variables in the heading of a for loop
• These variables are local to the for-loop
• They may be reused in other loops

```plaintext
int count = 1;
for (int i = 0; i < 10; i++) {
    count *= 2;
}
//using 'i' here generates a compiler error
```