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
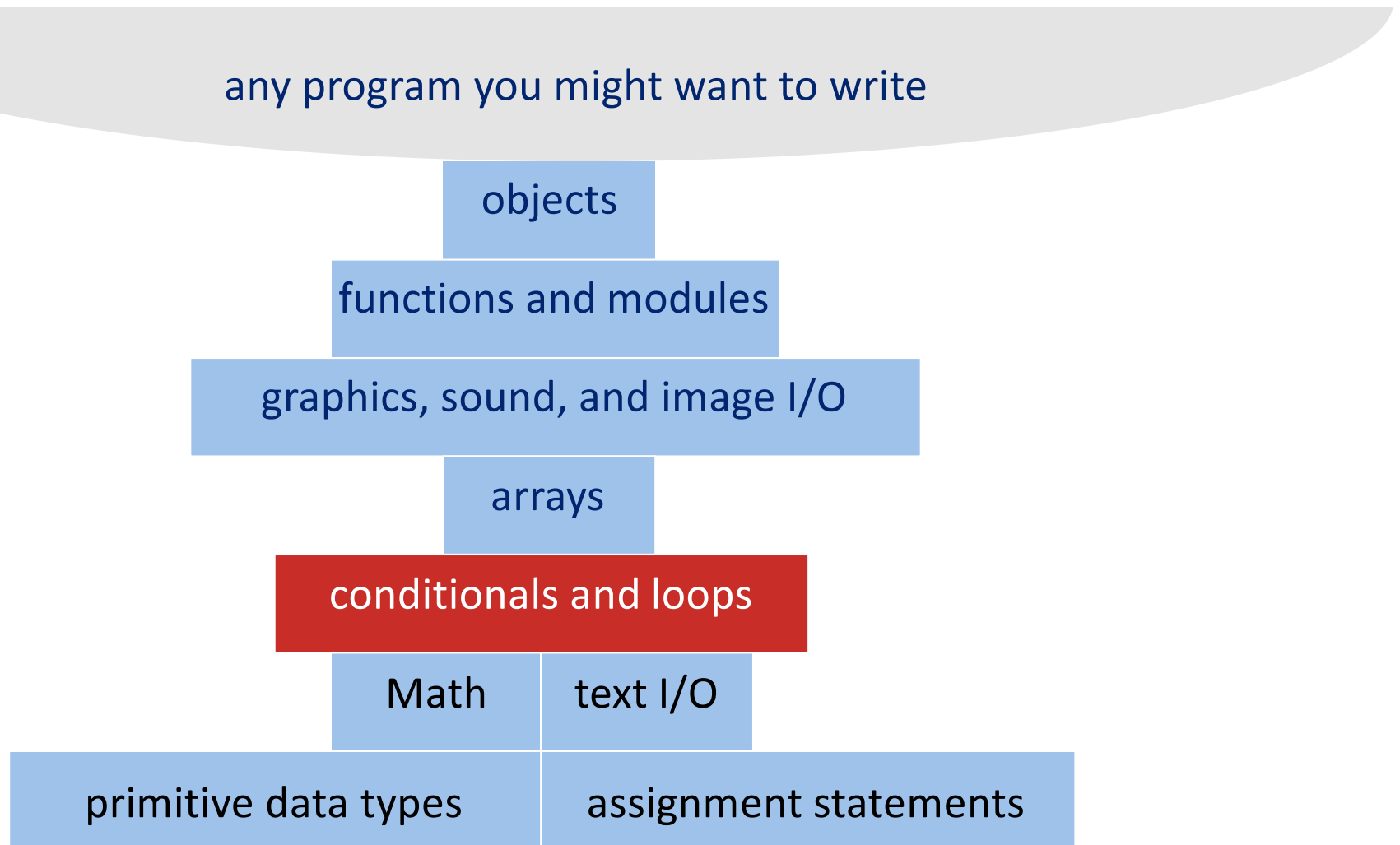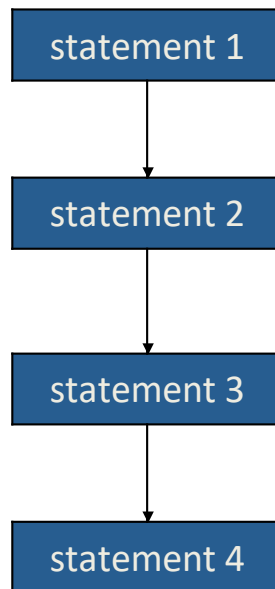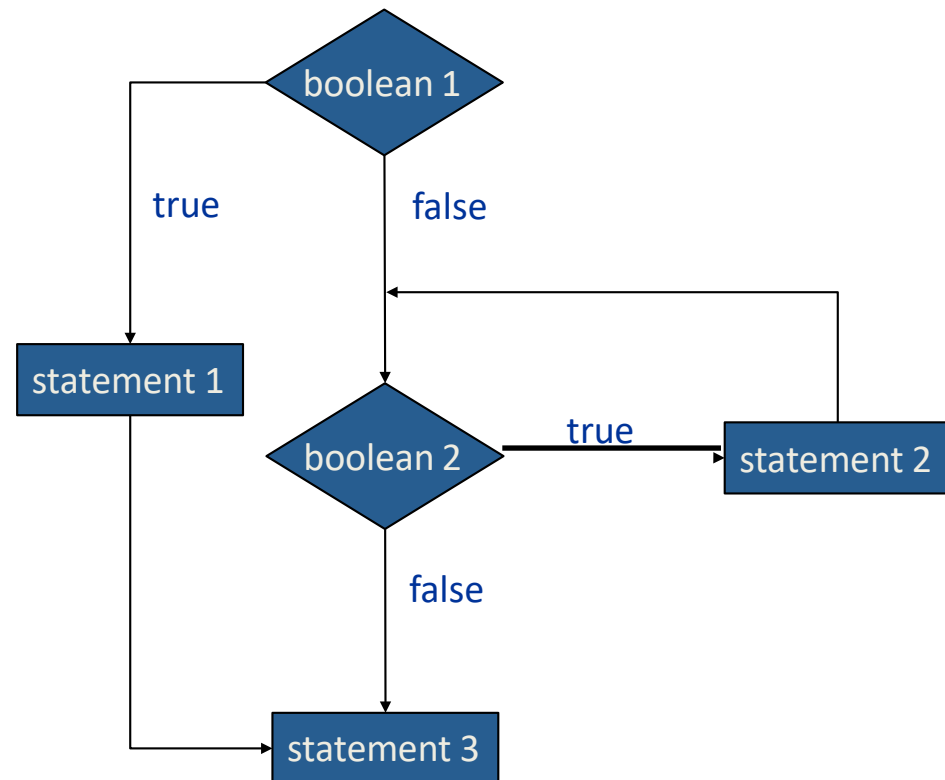
Control Flow

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

straight-line control flow
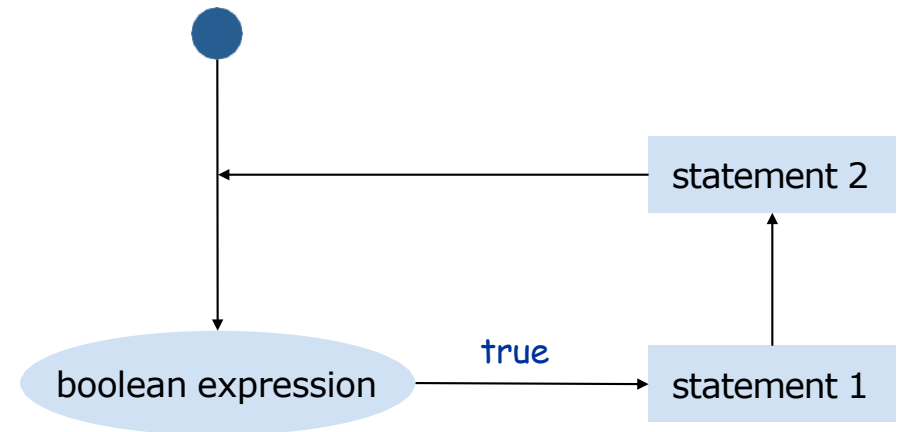
control flow with conditionals and loops

Animations with PennDraw

The Infinite While Loop

The infinite **while** loop: executes the loop body repeatedly

- Execute statement 1

- Execute statement 2

- …

- Repeat

```
while (true) {
    statement 1;
    statement 2;
}
```

loop body

statement 2

boolean expression → true → statement 1

Penn
Engineering

What will this do?

```
System.out.print("Program running");
while (true) {
    System.out.print(".");
}
System.out.println();
System.out.println("Program Exiting");
```

Using PennDraw for animation

- PennDraw.enableAnimation(10)

  Animation at 10 frames per second

- PennDraw.advance()

- PennDraw.disableAnimation()

Using PennDraw for Animation

```
public static void main(String[] args) {
    PennDraw.setCanvasSize(500, 500);
    PennDraw.enableAnimation(30);

    while (true) {    // repeats forever
        // draw frame of animation (your code here)

        PennDraw.advance();  // display next frame

    }
}
```

In-Class Demo: Bouncing Ball
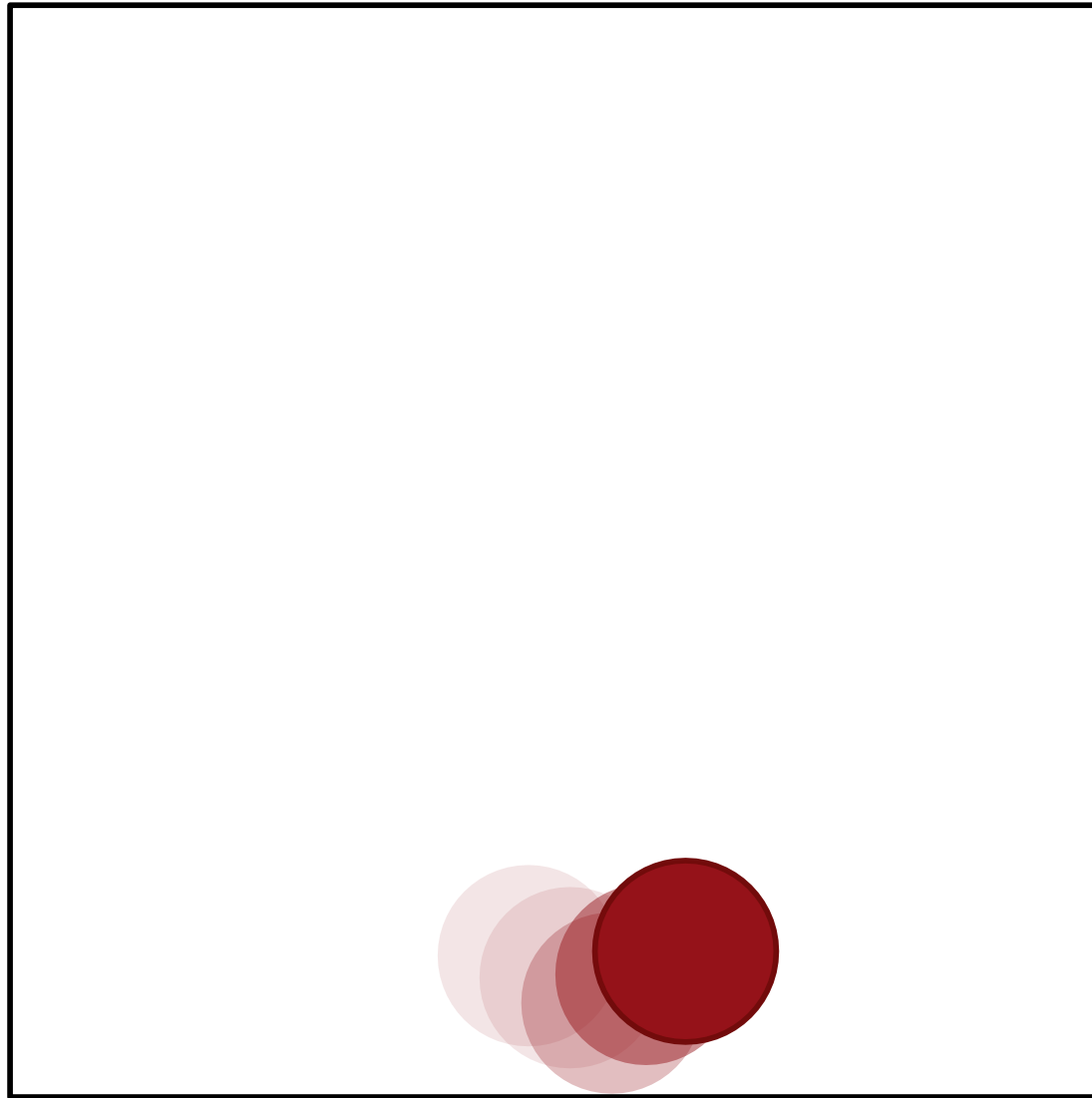
Equations of Motion (Simplified)

s = displacement
t = time
v = velocity
a = acceleration

- Constant acceleration (a)

$$s_{i+1} = s_i + v_i \, \Delta t$$
$$v_{i+1} = v_i + a \, \Delta t$$

In-Class Demo: Bouncing Ball
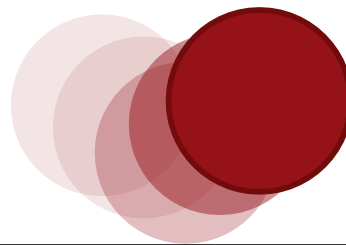
Conditionals

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

boolean expression

true   false

statement T   statement F
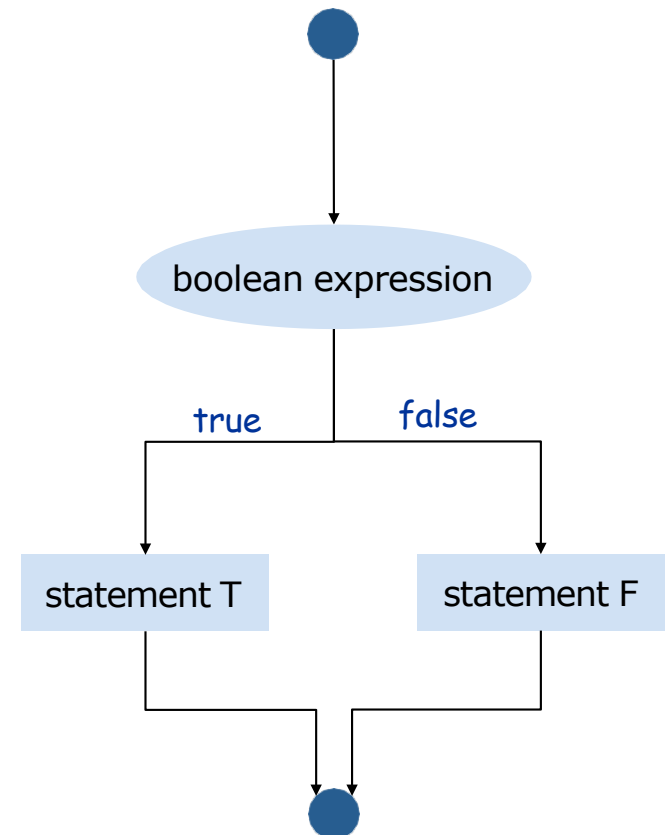
How could we write a program to model this parking sign?

- How do we provide a number to java?

- Command-line arguments
  - args[0] is the first argument, args[1] the second argument and so on
  - args[0] is a String

8.00 AM
To
3.00 PM

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 John
- want the program say "Hello John" as opposed to "Hello World"

Command line arguments

```java
public class Hello {
    public static void main (String[] args) {
        String name = args[0];
        System.out.println("Hello " + name);
    }
}
```

Command line arguments

- args[0] will be a String
- How to convert a String to an integer?
- Integer.parseInt()

Back to parking sign 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)
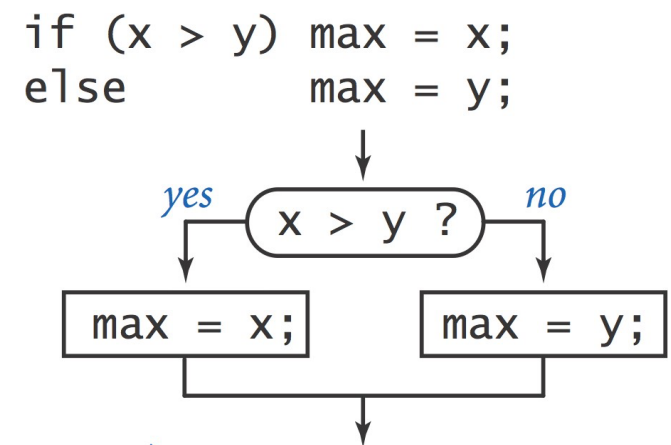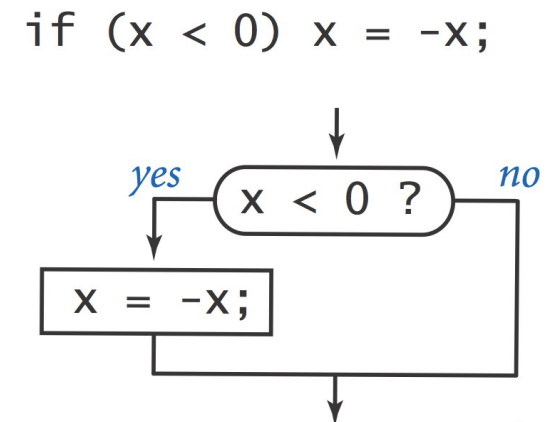
- true $\rightarrow$ false,   false $\rightarrow$ true

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 (x < 0) x = -x;
```



```
boolean
expression
          ↓
if ( x > y )
{
            int t = x;
sequence  → x = y;
   of       y = t;
statements
}
```

```
if (x > y) max = x;
else        max = y;
```


29
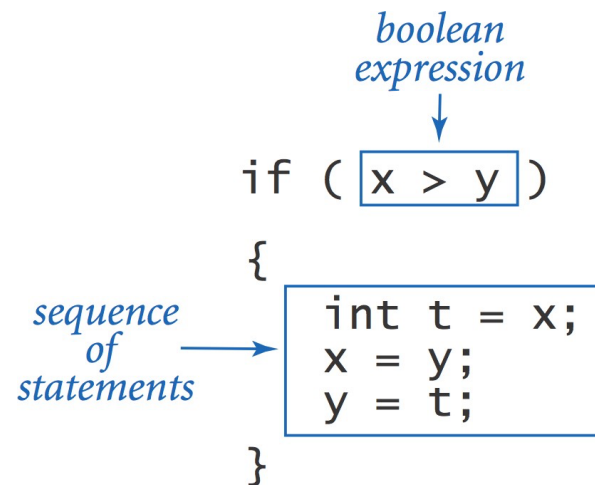
If Statement

- 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
```

Penn Engineering

26

If Statement Examples

absolute value	`if (x < 0) x = -x;`
put x and y into sorted order	```if (x > y)
{
int t = x;
x = y;
y = t;
}```
maximum of x and y	```if (x > y) max = x;
else   max = y;```
error check for division operation	```if (den == 0) System.out.println("Division by zero");
else   System.out.println("Quotient = " + num/den);```
error check for quadratic formula	```double discriminant = b*b - 4.0*c;
if (discriminant < 0.0)
{
System.out.println("No real roots");
}
else
{
System.out.println((-b + Math.sqrt(discriminant))/2.0);
System.out.println((-b - Math.sqrt(discriminant))/2.0);
}```

Penn Engineering

Conditionals: if-else-if-statement

```
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:

Income	Rate
0 – 47,450	22%
47,450 – 114,650	25%
114,650 – 174,700	28%
174,700 – 311,950	33%
311,950 -	35%

5 mutually exclusive alternatives

Nested If Statements

Use nested **if** statements to handle multiple alternatives

```
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;
        }
    }
}
```

Income	Rate
0 - 47,450	22%
47,450 – 114,650	25%
114,650 – 174,700	28%
174,700 – 311,950	33%
311,950 -	35%

Penn Engineering

Nested If Statements

Income	Rate
0 – 47,450	22%
47,450 – 114,650	25%
114,650 – 174,700	28%
174,700 – 311,950	33%
311,950 -	35%

5 mutually exclusive alternatives

Alternative shortened version:

```
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?

Income	Rate
$0 - 47,450$	22%
$47,450 - 114,650$	25%
$114,650 - 174,700$	28%
$174,700 - 311,950$	33%
$311,950 -$	35%

5 mutually exclusive alternatives

```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;
```

Conditionals: switch-statement

- Works like a if-else statement
- Convenient for large numbers of value tests

```
switch( expression ) {
    case label1:            // label1 equals expression
        statements;
        break;
    case label2:            // label2 equals expression
        statements;
        break;
    default:                // Nothing matches
        statements;
}
```

```java
public static void main(String[] args) {
    int month = Integer.parseInt(args[0]);
    String monthString;
    switch (month) {
        case 1:  monthString = "January";
                 break;
        case 2:  monthString = "February";
                 break;
        case 3:  monthString = "March";
                 break;
        case 4:  monthString = "April";
                 break;
        case 5:  monthString = "May";
                 break;
        case 6:  monthString = "June";
                 break;
        case 7:  monthString = "July";
                 break;
        case 8:  monthString = "August";
                 break;
        case 9:  monthString = "September";
                 break;
        case 10: monthString = "October";
                 break;
        case 11: monthString = "November";
                 break;
        case 12: monthString = "December";
                 break;
        default: monthString = "Invalid month";
                 break;
    }
    System.out.println(monthString);
}
```

An aside ... Operators

+, -, *, / and ...

```
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 repeated multiple times.

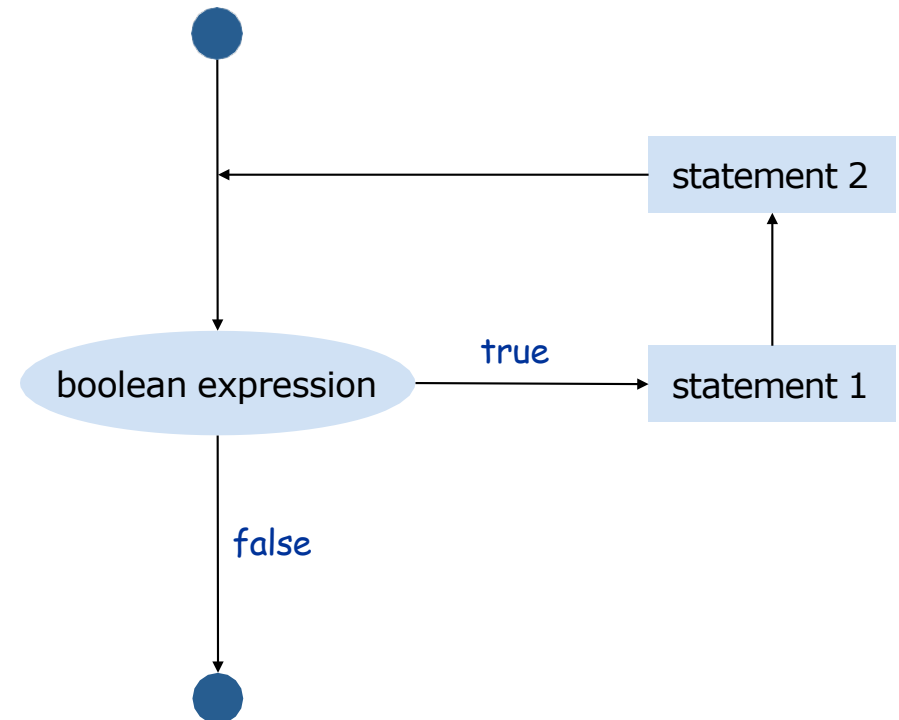Options

- The while–loop
- The for–loop

The While Loop

While Loop

The **while** loop: A common repetition structure
- Evaluate a **boolean** expression
- If **true**, execute some statements
- Repeat

loop continuation condition

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

true

boolean expression ⟶ statement 1

statement 2

false

The Infinite While Loop, Re-examined

```java
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:

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

N = 4

i	v	i <= N
0	1	true
1	2	true
2	4	true
3	8	true
4	16	true
5	32	false

Penn Engineering

42

While Loop Challenge

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

```
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?

```
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 ;           ⟵   initialization of loop control variable


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

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();

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

The For Loop

54
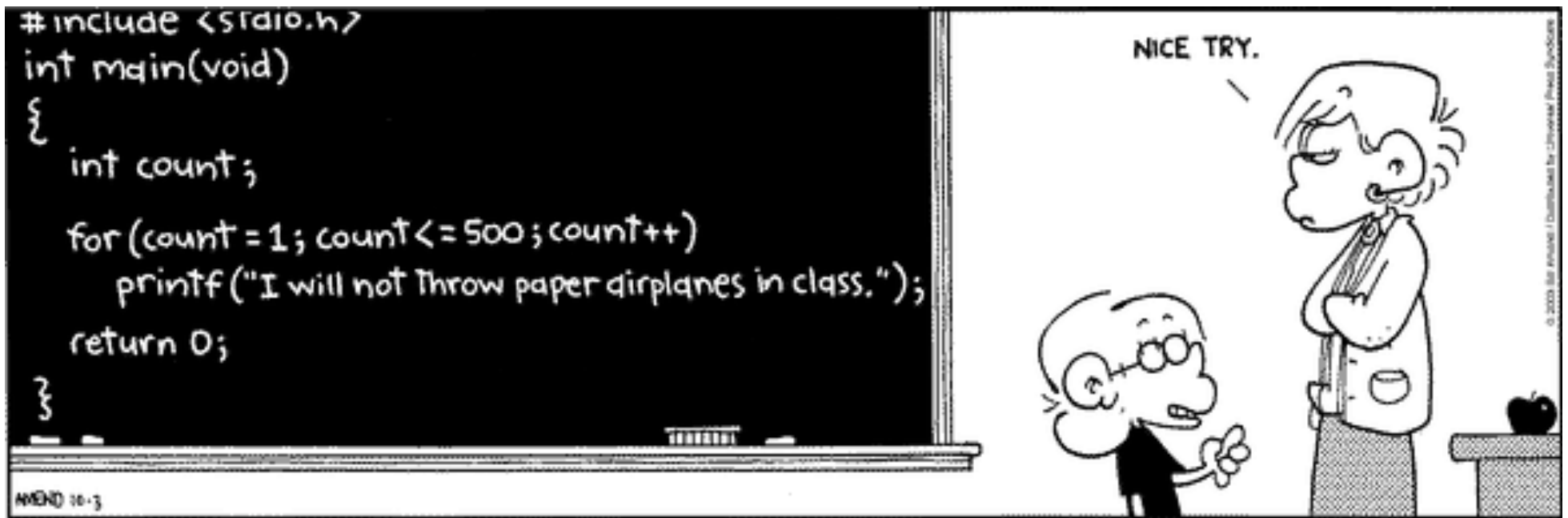
For Loops

- 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

for (int i = 1; i < 101; i = i + 1) {

     ↑   ↑   ↑

   initialization   modification

}   test

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

```java
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

v	i	i <= N
1	0	true
2	1	true
4	2	true
8	3	true
16	4	true
32	5	false

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 2s:

```
// modify part can be "i  -= 2"
for ( i = 10;  i > 0;  i = i - 2 ) {
   System.out.println( i ) ;
}
```

For loop examples

| compute a finite sum (1 + 2 + …+ N) | ```
int sum = 0;
for (int i = 0; i <= N; i++)
    sum += i;
System.out.println(sum);
``` |
| print largest power of two less than or equal to N | ```
int v = 0 ;
for (v = 1; v <= N/2; v *= 2);
System.out.println(v);
``` |
| | |

When Does a *for* Loop Initialize, Test and Modify?

- Just as with a **while** loop, 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
- The **for** loop is easier to write and read for counter-controlled loops.

```
public static void main(String[] args){
   PennDraw.setCanvasSize(500,500);

   double radius = 250.0;
   while ( radius > 1.0 ) {
       PennDraw.circle(0.5, 0.5, radius / 500);
       radius = radius - 5.0;
   }
}
```

```
public static void main(String[] args){
   PennDraw.setCanvasSize(500,500);

   for ( double radius = 250.0; radius > 1.0; radius -= 5.0 ) {
       PennDraw.circle(0.5, 0.5, radius / 500);
   }
}
```

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??        OUTPUT:
…
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

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