Announcements & Reminders

- HW00 was due on Wednesday, last late day for submission is today
- HW01 is due on Wednesday, September 18
- Check-in Assignment #1 is due on Monday, September 16 before the start of class
    - Complete on Codio, submit on Gradescope
    - Takes ~10 minutes
    - Purpose is **low-stakes, regular** practice on previous week's materials

Interactivity & Animation in PennDraw

Overview

Like humans, programs should be able to repeat some actions while a condition is true

In this module we will learn how to express repetitions in a program!

Example:

- while *hungry is true* eat; when *hungry is false* stop eating

Learning Objectives

- Write an animation loop in PennDraw
- Create drawings that change over time
- Write programs that respond to interaction from the user

Iteration

- Repetition of a program block while a condition is true
- Iteration allows us to control the flow of a program like conditionals
    - Instead of "do or not do", the question is "how long to repeatedly do"

Two options:

- while loop
    - introduced today, expanded upon on Friday
- for loop
    - introduced on Friday

while loop

Executes the body of the loop as long as (or while) a Boolean expression is `true`

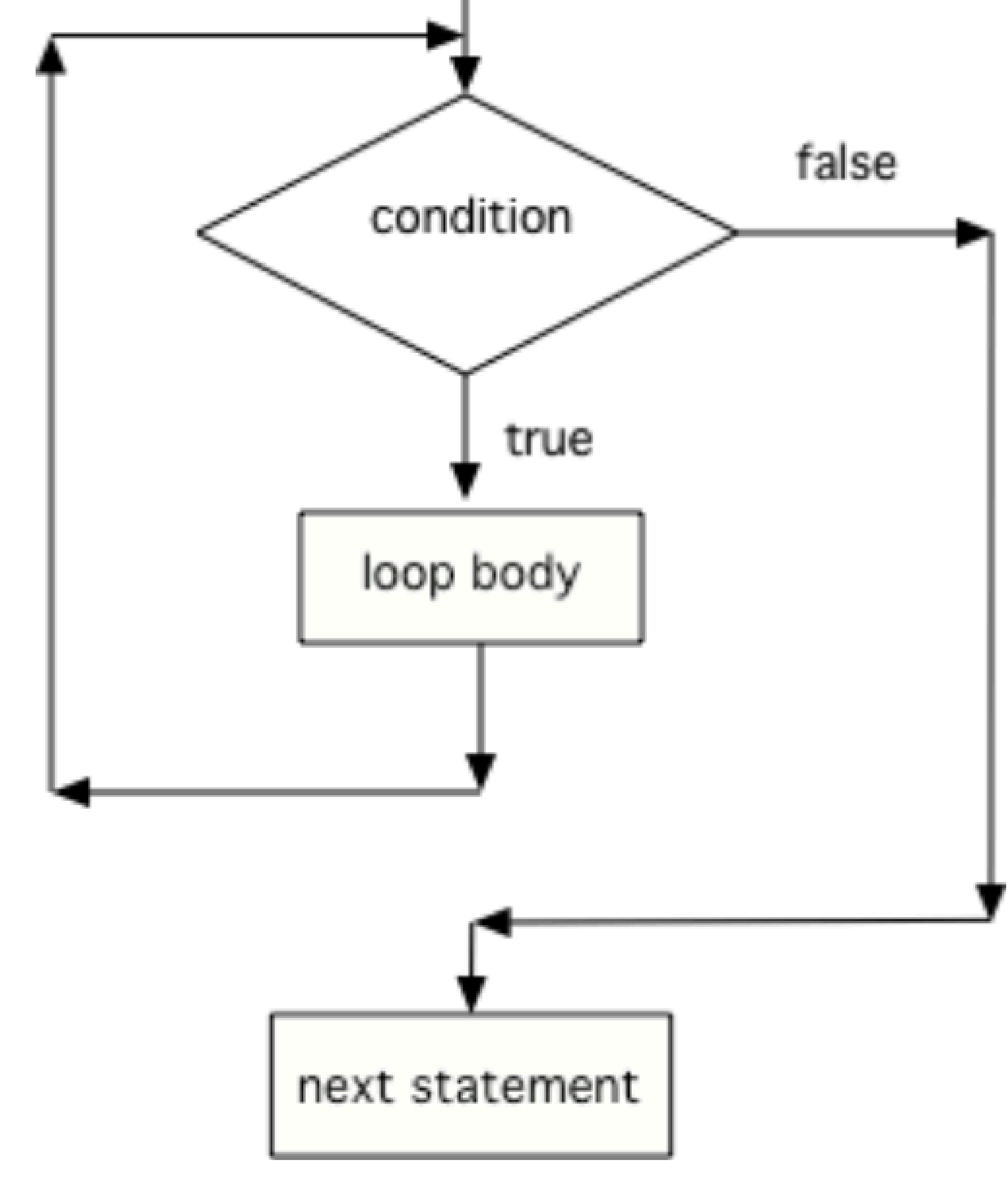

Figure 3: Control Flow in a while Loop

The simplest while loop

```
while (true) {
        // start of the loop
        statements;
        statements;
        statements;
        statements;
        // end of the loop
}
// code here won’t get run!
```

Counting to Infinity

A program that uses a while loop to repeatedly increment the value of a variable.

```
public class CountingUp {
    public static void main(String[] args) {
        int counter = 0; // initialize the variable outside the loop
        while (true) {
            System.out.println(counter);
            counter = counter + 1; // increment our counter after printing
        }
    }
}
```

Animation & Frames

- Animation (in film, TV, or computer graphics) is acheived by showing a rapid sequence of discrete images.
  - Each distinct image is called a "frame"
  - Showing ~24 frames per second leads to the illusion of smooth, continuous motion.
- We can create animations in PennDraw by drawing many frames per second
  - Use a loop to do the repeated drawing—one iteration draws one frame
  - Change values of variables in the loop body to make the frames change with each iteration

Basic Recipe for Animation with PennDraw

Setup

- `PennDraw.setCanvasSize`, `PennDraw.enableAnimation`, variable declarations

The `while(true)` loop

- Clear the screen, then draw the next frame
- Update the values of variables used in drawing
- PennDraw.advance()

SlidingSquare.java as a Template

```
double xCenter = 0;
double yCenter = 0.5;
double sideLength = 0.1;
PennDraw.setCanvasSize(400, 400);
PennDraw.enableAnimation(30);
while(true) {
    PennDraw.clear(); // clear the previous frame
    PennDraw.square(xCenter, yCenter, sideLength); // draw the new frame
    xCenter = xCenter + 0.001; // update the variable used in drawing
    if (xCenter > 1.1) {
        xCenter = 0; // if the square would be drawn off the screen, reset
    }
    PennDraw.advance();
}
```

(note: class declaration & main omitted for space)

Adding in Interaction: Mouse

User mouse clicks and mouse position can be monitored using PennDraw

Function	Return Type	Description
`PennDraw.mousePressed()`	`boolean`	Returns `true` if the mouse is being held this frame.
`PennDraw.mouseX()`	`double`	Returns the **x coordinate** of the mouse's current location, e.g. `0.9` or `0.1443`
`PennDraw.mouseY()`	`double`	Returns the **y coordinate** of the mouse's current location, e.g. `0.9` or `0.1443`

Click Counter

```java
public class ClickCounter {
    public static void main (String[] args) {
        int numberOfClicks = 0;
        PennDraw.enableAnimation(30);
        while (true) {
            PennDraw.text(0.5, 0.5, "Number of Clicks: " + numberOfClicks);
            if (PennDraw.mousePressed()) {
                numberOfClicks += 1;
            }
            PennDraw.advance();
        }
    }
}
```

Adding in Interaction: Keyboard

User key presses can also be registered!

Function	Return Type	Description
`PennDraw.hasNextKeyTyped()`	`boolean`	Returns `true` if there is an unread key press
`PennDraw.nextKeyTyped()`	`char`	Returns the next unread key typed and clears it from the queue

🚨 Never use `nextKeyTyped()` without checking `hasNextKeyTyped()` first!! 🚨

```java
public class LightSwitch {
    public static void main (String[] args) {
        boolean on = false;
        PennDraw.enableAnimation(30);
        while (true) {
            if (on) {
                PennDraw.clear(PennDraw.BLACK);
            } else {
                PennDraw.clear(PennDraw.YELLOW);
            }
            if (PennDraw.hasNextKeyTyped()) {
                char c = PennDraw.nextKeyTyped();
                if (c == 'x') {
                    on = !on;
                }
            }
        }
    }
}
```

Randomness

Predictability is overrated—let's explore how we can get our programs to behave in random ways.

- `Math.random()` is a function that returns a `double` value between `0` and `0.999...`.
    - Never `1`!
    - The randomness is *uniform*: each value in the output range is equally likely.

Flip a Coin

```
public class FlipACoin {
    public static void main(String[] args) {
        double randomNumber = Math.random();
        boolean isHeads = randomNumber > 0.5; // this will be true 50% of the time!
        if (isHeads) {
            System.out.println("Heads, I win!");
        } else {
            System.out.println("Tails, you lose!");
        }

    }
}
```

Random Events

- Each call to `Math.random()` gives a result between `0` and `1` where each is equally likely.
- Therefore, there's a 100% chance the number generated is less than `1`
- There's a 90% chance the number generated is less than `0.9`
- There's an 80% chance the number generated is less than `0.8`
- There's an 53.4% chance the number generated is less than `0.534`

➡️ to simulate an event that happens $x\%$ of the time, draw a random number and check if it falls in the range of $(0, \frac{x}{100}]$

Generate a Random Integer

- We can expand the range of random outputs by multiplying by the width of the desired range
  - `Math.random() * n` will be a random `double` between `0` and `n` (but not `n` itself).
  - `Math.random() * 10` might be `3.43`, `0.0342`, `9.99991`, etc.
- We can limit of possible outputs to `int` values by *casting*.
  - `(int) (Math.random() * n)` throws away the decimal part of `Math.random() * n` and gives an `int` value.
  - `(int) 3.43` becomes `3`, `(int) 9.99991` becomes `9`
  - The possible values returned from `(int) (Math.random() * n)` are `0, 1, 2, 3, ... n-1`

Pick a Random Color

```
int red = (int) (Math.random() * 256);
int green = (int) (Math.random() * 256);
int blue = (int) (Math.random() * 256);
PennDraw.setPenColor(red, green, blue);
```

Common Misconceptions about `Math.random()`

- Misconception: each call to `Math.random()` produces a the same value
  - Correction: each time we write `Math.random()`, it will evaluate to a new random `double`.
- Misconception: storing the result of `Math.random()` in a variable means we can't know what value the variable has
  - Correction: we can print out the variable and unless we manually reassign it, the variable will always have the same value
- Misconception: `(int) Math.random() * n` gives a random `int` between `0` and `n-1`
  - Correction: parentheses are needed! `(int) (Math.random() * n)` is correct. Otherwise, the value is `0` always.

Iteration

Loop control variable

The loop condition involves a **loop control variable**

The **loop control variable** controls when the loop stops!

The loop condition tests that the value of the loop control variable matches a specific condition (`>, <, >=, <=, ==, !=`)

Three steps of a while loop

1. Initialize the loop variable (before the while loop)
2. Test the loop variable (in the loop header)
3. Change the loop variable (in the while loop body at the end)

```
int count = 1;

while(count <= 10)
{
    System.out.println(count);
    count++;
}
```


Figure 4: Three Steps of Writing a Loop

Tracing a while loop

Evaluate a Boolean expression

- If **true:**
  - Execute the body of the loop
  - Repeat
- If **false**, exit the loop

Tracing a while loop

```
int count = 1;
while (count <= 5) {
    System.out.println(count);
    count++;
}
```

count	Count <= 5	Output
1	true	1
2	true	2
3	true	3
4	true	4
5	true	5
6	false	

Infinite loops!

```
int count = 1;
while (count <= 5) {
    System.out.println(count);
    // missing loop control variable update step!
}
```

```
int count = 1;
while (count <= 5) {
    System.out.println(count);
    count--; //update step doesn't bring us closer to making condition false.
}
```

Off-by-one errors

Off-by-one errors happen when the loop runs one too many/too few times!

- Happens when we use the incorrect relational operator in the test loop step

```
// count from 1 to 5?
int count = 1;
while (count < 5) {
    System.out.println(count);
    count++;
}
```

The loop will execute 4 times instead of 5 because we used `<` instead of `<=`!

Aside: Scope

Scope is the part of the program where a variable exists

- A variable’s scope is within the pair of curly braces `{}` it is defined in
- Variables declared in a loop or `if`/`else if`/`else` only exist inside that structure.
- A variable declared in a loop is "reset" on each iteration of the loop
- Variables declared in separate scopes can have the same name.

Scope Example

A variable declared in a loop is “reset” on each iteration of the loop

```
int x = 0;
while (x < 10) {
    int y = 0;
    System.out.println(y);  // always prints 0, y stops existing on each iteration
    y++;
    x++;
}
System.out.println(x);  // prints 10.
// trying to print y here would get a compiler error
```

`for` loop

Used when you know how many times you want the loop to execute

```
for (initialization; condition; change) {
    // loop body statements go here
}
```

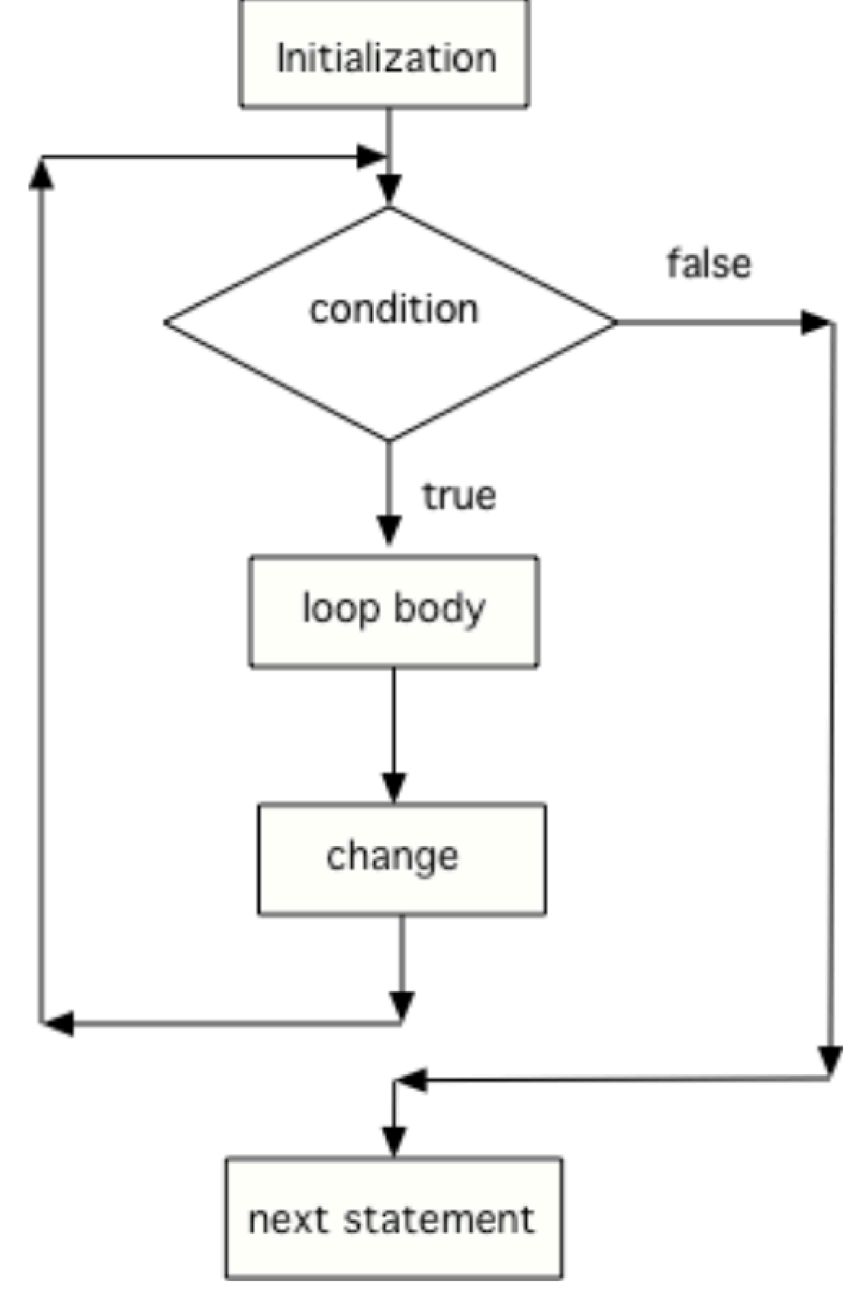

Figure 2: Control flow in a for loop

`for` vs `while` Loops

These loops are equivalent!

```
for (int x = 3; x > 0; x--) {
    System.out.println(x);
}
```

```
int x = 3;
while (x > 0) {
    System.out.println(x);
    x--;
}
```

`for` *Loop Patterns*

- Both `for` and `while` can be used interchangeably, but sometimes one makes more sense over the other.
- Prefer the `for` loop for counting fixed numbers of iterations!
  - Here are two `for` loops that execute some statements `10` times.

```java
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

```java
for (int i = 1; i <= 10; i++) {
    System.out.println(i);
}
```

`while` *Loop Patterns*

- Both `for` and `while` can be used interchangeably, but sometimes one makes more sense over the other.
- Use `while` when # of iterations is indeterminate or depends on multiple vars

```
boolean hasFoodLeft = true;
boolean isHungry = true;
while (hasFoodLeft && isHungry) {
    // represent taking a bite
    numBites++;
    hunger--;
    // update loop control variables
    isHungry = hunger > 0;
    hasFoodLeft = numBites < mealSize;
}
```

Loops and Strings

Loops are often used for **String Traversals** or **String Processing**.

- Traversing a string involves going through a string character by character
- Characters are located based on their position (or index) in the string

The first character in a Java String is at index `0` and the last character is at `length() - 1`

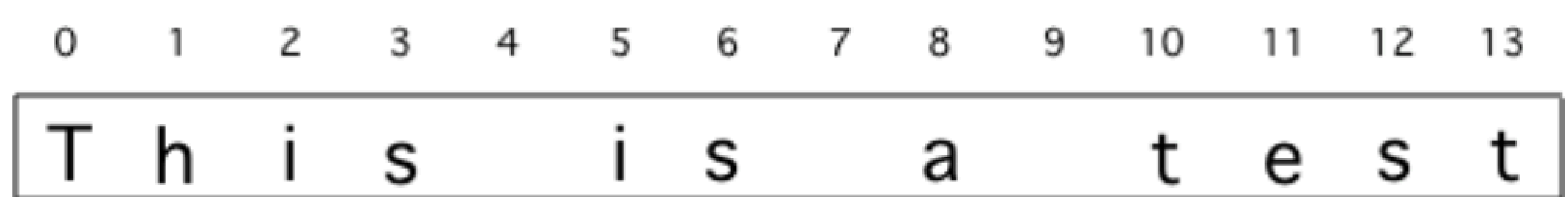

Figure 1: A string with the position (index) shown above each character

Loops and Strings

```
String s = "welcome";
int count = 0;
while (count < s.length()) {
  System.out.println(s.charAt(count));
  count++;
}
```

```
String s = "welcome";
for (int i = 0; i < s.length(); i++) {
   System.out.println(s.charAt(i)):
}
```

Both programs will print the characters in s one at a time—`for` loop looks cleaner!

Nested Loops

Nesting happens when a loop is contained inside another one!

- In each iteration of the outer loop, the inner loop will be re-started.
- The inner loop must finish all of its iterations before the outer loop can continue to its next iteration

```
for (int row = 1; row <= 3; row++)
{
    for (int col = 1; col <= 5; col++)
    {
        System.out.print("*");
    }
    System.out.println();
}
```


Figure 1: Nested Loops

Reasoning Through Nested Loops

Think abstractly about the purpose of the loops—going inside out often helps!

- Inner Loop ↔ "Print 5 Stars"
- Outer Loop ↔ ("Print 5 Stars", then print an empty line) x3


```
for (int row = 1; row <= 3; row++)
{
    for (int col = 1; col <= 5; col++)
    {
        System.out.print("*");
    }
    System.out.println();
}
```

Figure 1: Nested Loops