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.
The simplest while loop

```java
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.

```java
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 achieved 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

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PennDraw.mousePressed()</td>
<td>boolean</td>
<td>Returns true if the mouse is being held this frame.</td>
</tr>
<tr>
<td>PennDraw.mouseX()</td>
<td>double</td>
<td>Returns the x coordinate of the mouse's current location, e.g. 0.9 or 0.1443</td>
</tr>
<tr>
<td>PennDraw.mouseY()</td>
<td>double</td>
<td>Returns the y coordinate of the mouse's current location, e.g. 0.9 or 0.1443</td>
</tr>
</tbody>
</table>
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!

<table>
<thead>
<tr>
<th>Function</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PennDraw.hasNextKeyTyped()</td>
<td>boolean</td>
<td>Returns true if there is an unread key press</td>
</tr>
<tr>
<td>PennDraw.nextKeyTyped()</td>
<td>char</td>
<td>Returns the next unread key typed and clears it from the queue</td>
</tr>
</tbody>
</table>

Never use `nextKeyTyped()` without checking `hasNextKeyTyped()` first!!
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 $0.999\ldots$.
  - Never $1$!
  - The randomness is *uniform*: each value in the output range is equally likely.
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!");
        }
    }
}
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 $1, 2, 3, \ldots, n-1$
Pick a Random Color

```java
int red = (int) (Math.random() * 255);
int green = (int) (Math.random() * 255);
int blue = (int) (Math.random() * 255);
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.