Multimedia Programming for Arts and Design

Week 1 (Mon & Wed)
IAT-265, Summer 2011
School of Interactive Arts and Technology

This week's topics

- About the course
- Multimedia as a context
- Java and Processing
- Drawing primitives
- Method, parameter, argument
- Variables of Primitive Types
- Animation with setup() & draw()

About the course

- Introduce concepts such as ***variables, data types, conditionals, loops,*** *and* ***OOP concepts*** in the context of multimedia processing

- Enjoy writing programs to ***draw graphics, modify pictures, create animations, support interactivities, …***

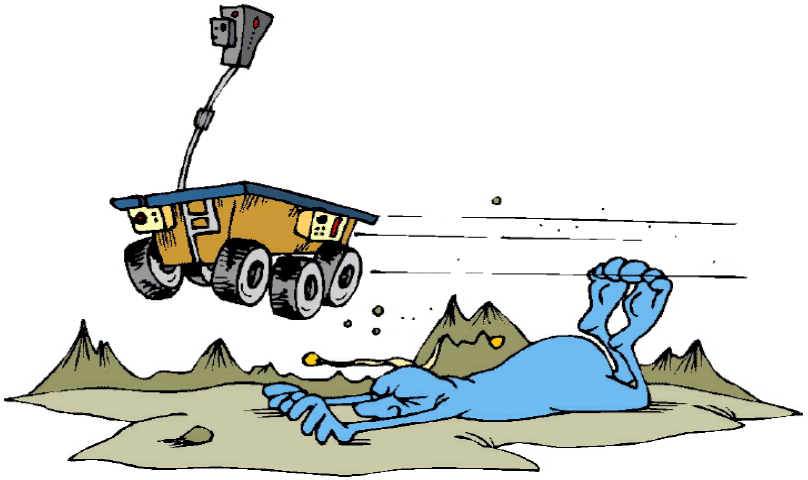
Syllabus

- Processing
  - Computer graphics
  - Animations
  - Images
  - Interactions
- Java
  - Variables, control structures
  - Methods
  - Classes/Objects
    - Encapsulation/Inheritance/ Polymorphism
- Computer graphics
  - Points, lines, curves, shapes, polygons
  - Transformations
  - Object-oriented graphics
- Recursion
- Basic data structures

Why Learn to Program?

- If you can imagine it, you can make it “real” on a computer

- Computers will continue to have a major impact on modern life
  - Movies, games, business, healthcare, science, education, etc

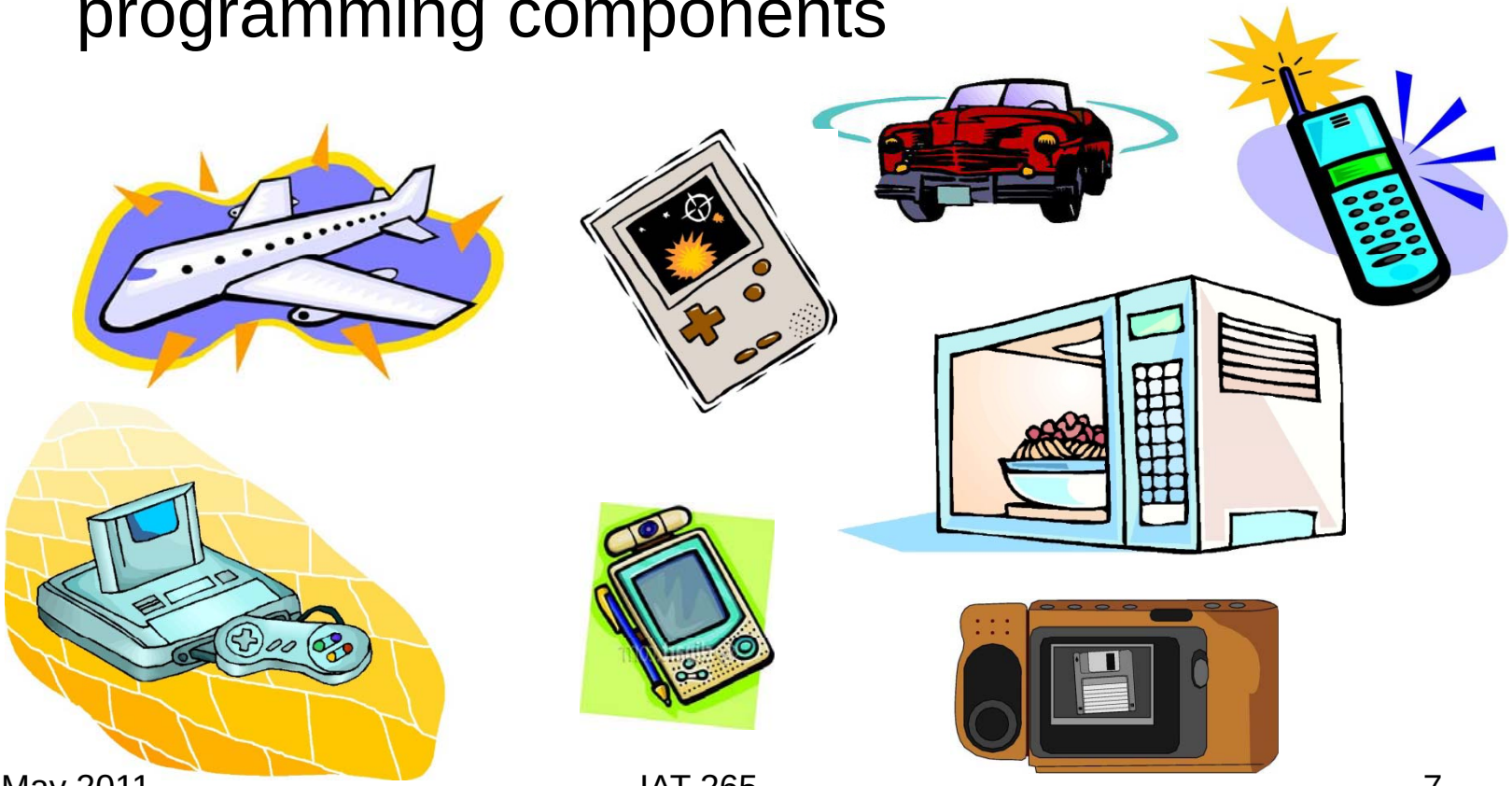


But why not just use existing applications?

- See what some professionals say
  - Barbara – Biological visualization
    - "*Being able to look at a problem, and come up with a computer solution, is a valuable skill in any branch of science*"
  - Siobhan – Google Gmail
    - "*It's really exciting, you have the power to create anything. You have the opportunity to make a big difference in the world*"

Programming Support Interaction Design

- All interactive devices/applications have programming components

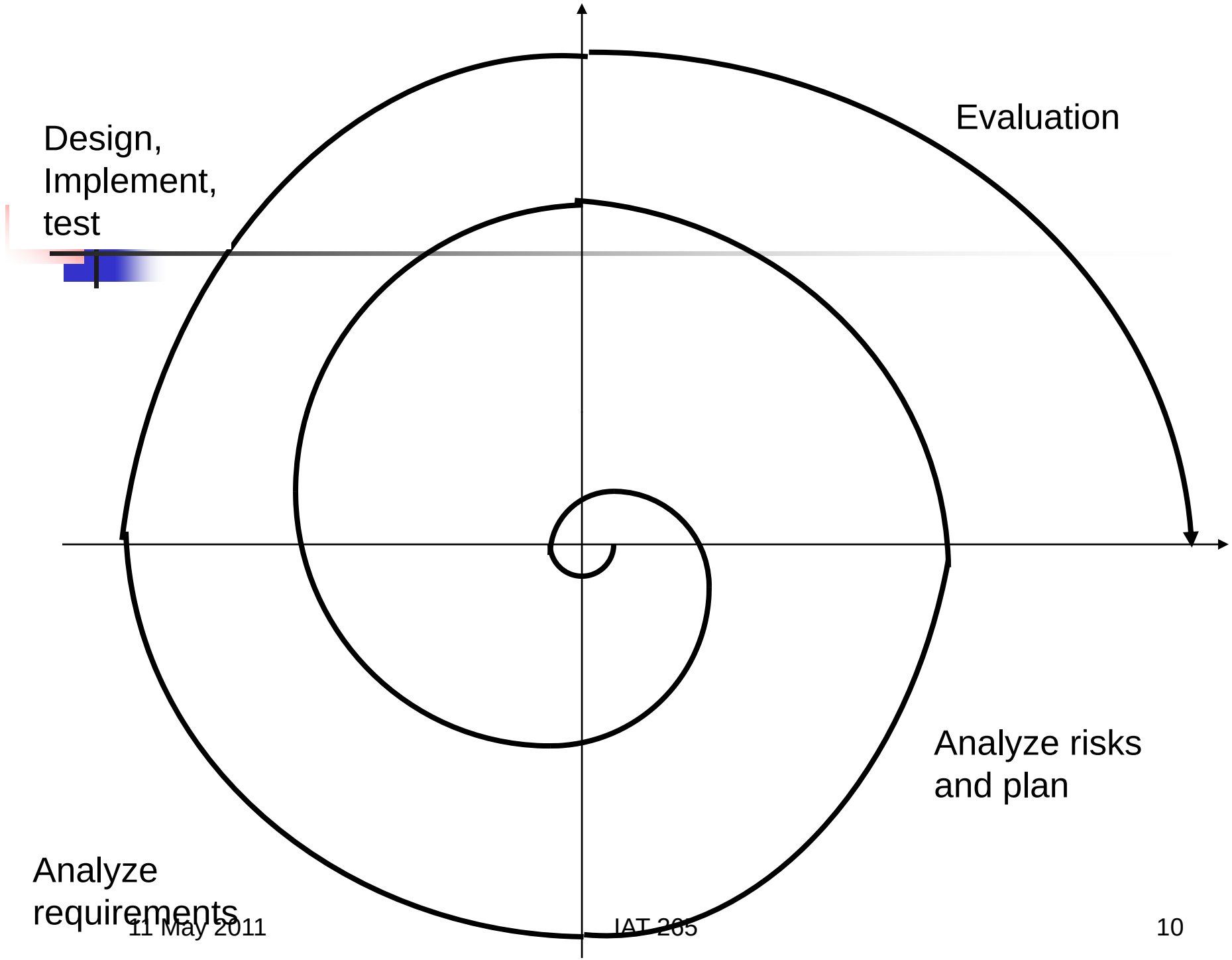


Programming is challenging

- Beginners have a hard time understanding some of the core concepts
  - Expressions with more than two or more items
    - if (a < b) is okay
    - if (a < b && c > d) is hard
    - if (a < b && c > d)… else if (a < b && c < d) is daunting
  - Iterations (loops)
  - Difference between ‘define a function’ and ‘call a function’
  - Class/Object????
- Beginners have a hard time putting statements together to accomplish a task

But
there are ways to conquer it …

- Iterative development (small pieces at a time that rumps up complexity finally)
  - This process is called Spiral Model in software engineering
  - That's why we let you do five assignments that will ramp up complexity gradually
  - This also means when you do your coding, you should always: code a little bit, test a little bit, and have working code!!


Design,
Implement,
test
Evaluation
Analyze risks
and plan
Analyze
requirements

Grading

- Required parts
  - Homework Assignments 40%
  - Quizzes 20%
  - Final exam 40%

Requirements and Expectations

- Attend all lectures and labs
  - Slides and coding demos are put in webct after class
- Do all the ASSIGNMENTS!!
- Do reviews on a WEEKLY basis
  - Read slides and readings (listed in Syllabus), and try to understand the basic concepts
  - Attend instructor/TAs' office hours for clarification
- Practice, Practice, Practice!!
  - Programming courses are always challenging and need practice

Write programs and have FUN!!

- Asteroids with GUI
- Monsteroids
- Osmos Clone
- http://www.youtube.com/watch?v=jMsOTiLElcU&feature=player_embedded
- http://www.youtube.com/watch?v=ySTm5ebrFlM&feature=player_embedded
- Some examples from last semester

Multimedia as a Context

- What is Multimedia?

Multimedia

- Applications that use multiple modalities, to their advantages, of **text, images, drawings (graphics), animation, video, sound**, and **interactivity**

  - Z. Li & M. Drew *Fundamentals of Multimedia*

- **Multimedia** (Lat. Multum + Medium) is media that uses multiple forms of information content and information processing (e.g. **text, audio, graphics, animation, video, interactivity**) to inform or entertain the (user) audience

  - From *Wikipedia*, the free encyclopedia

Typical Multimedia Applications

- Games
- Virtual reality
- World Wide Web
- Digital image/video editing and production
- Video teleconferencing
- Multimedia courseware
- …

Java is Object-Oriented

- We live in a world full of objects
  - Images, cars, remote controls, televisions, employees, students, …
- The older languages are procedural
- OOP languages have the added capability to encapsulate objects' properties and functions into one container – ***class***
  - Instances of a class are called ***objects***

Object Oriented vs. Procedural Languages

Procedural (e.g. C)

- We create some data representing an image
- We write a *procedure* that can accept the data and draw the image

Object Oriented (e.g. Java)

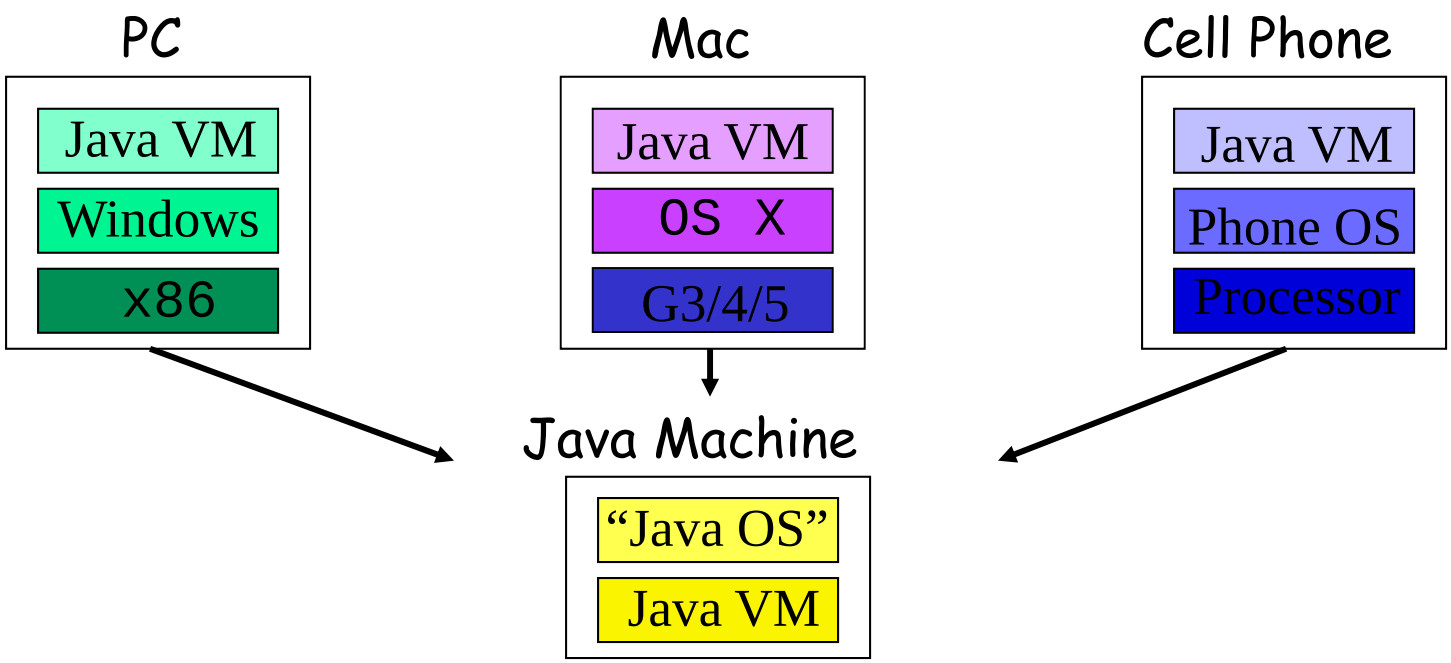
- We create a *class* that contains image data AND a procedure to draw it
- The data and the procedure (ability to draw) are in ONE "container“ – the class

Java

- Designers started with C++
  - Simpler
  - Safer
- Programming embedded systems
  - Toasters, microwave ovens, TV set top boxes
    - Reliability very important--avoid costly recalls
- Web programming
  - Incorporated into web browsers at critical moment

The virtual machine

- Since Java was designed to run on embedded systems, it was designed around a *virtual* machine
  - "Write once, run everywhere"



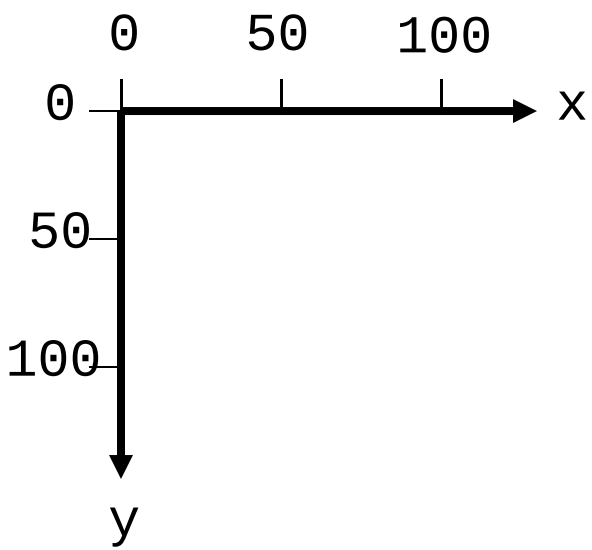
But we're using Processing

- Processing is built on top of Java
- Supports script-like coding
  - Easy to get simple programs up fast
  - But allows transition to full Java programming
- Has built-in methods and classes to make drawing easy
- Easy to export program to *applet*

Drawing in Processing

- Automatic creation of display window

- Window has a coordinate system for drawing



Let's draw a point: point()

`point(x,y)` – draws a point at the location x, y

Let's try it: `point(50, 50)`

Unexpected token: null – what ??

- Compiler errors appear in the bottom pane

All lines must be terminated with a semicolon ;

Drawing several points

```
point( 30, 20 );
point( 90, 20 );
point( 90, 80 );
point( 30, 80 );
```

Drawing more shapes

```
line(x1, y1, x2, y2);

triangle(x1, y1, x2, y2, x3, y3);

rect(x, y, width, height);
```

rectMode() – CORNER(default), CENTER, CORNERS,

```
ellipse(x, y, width, height);
```

ellipseMode() – CENTER (default), CORNER, RADIUS, CORNERS

Example: Draw shapes

- Try the following code in Processing, and see what you got

```
smooth();        //makes strokes smooth. You can try to
                 //remove it and see what happens
ellipse(60, 50, 85, 85);
rect(30, 20, 60, 60);
triangle(90, 20, 90, 80, 30, 80);
line(30, 20, 90, 80);
```

Controlling win size, color and line width

- Window's size specified by: size(width, height)

- Colors represented as Red Green Blue (RGB) values or Gray,
  - Each one ranges from 0 to 255

`background(R,G,B);` – set the background color

background(gray); – *gray*: 0 to 255 - specifies a value between black and white

`stroke(R,G,B);` – set the colors of the outline (default black)

stroke(gray)

`fill(R,G,B);` – set the fill color for shapes (default white)

fill(gray);

`strokeWeight(w);` – line width for outlines (default 1)

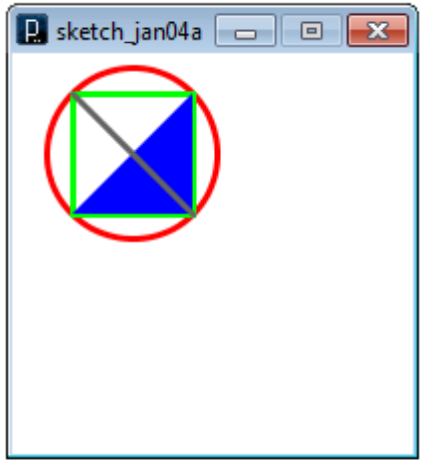
Disable Stroke and Fill

- Sometimes it is necessary to disable stroke or fill within your code:
  - noStroke(): – no outline drawing around shapes
  - noFill(): don’t fill the shapes (background shows through)

Example: Controlling color and line

- Add the following controlling method calls to your sketch:

```
size(200, 200);
background(255);
strokeWeight(3);
smooth();
stroke(255, 0, 0);
ellipse(60, 50, 85, 85);
stroke(0, 255, 0);
rect(30, 20, 60, 60);
fill(0, 0, 255);
noStroke();
triangle(90, 20, 90, 80, 30, 80);
stroke(100);
line(30, 20, 90, 80);
```



Comments

- Comments are non-program text you put in the file to describe to others (and yourself) what you're doing
- Important for being able to look back at your code and understand it
- Single-line comments begin with `//`
- Multi-line comments begin with `/*` and end with `*/`

Commenting and uncommenting lines is useful for figuring out code

Drawing arcs

arc(x, y, width, height, start, stop);

x, y – coordinates of the arc's ellipse

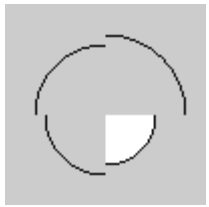
width, height – width, height of arc's ellipse

start, stop – angles to start, stop the arc (in radians, positive clockwise)

arc's ellipse may be changed with the ellipseMode()

- Example:

```
arc(50, 55, 50, 50, 0, PI/2);
noFill();
arc(50, 55, 60, 60, PI/2, PI);
arc(50, 55, 70, 70, -PI, -PI/2);
arc(50, 55, 80, 80, -PI/2, 0);
```



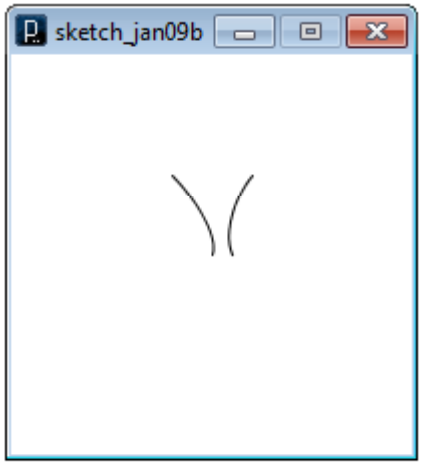
Drawing curves

curve(*cpx1*, *cpy1*, *x1*, *y1*, *x2*, *y2*, *cpx2*, *cpy2*);

cpx1, *cpy1* – coordinates for the beginning control point
x1, y1 – coordinates of the curve's starting point
x2, y2 – coordinates of the curve's ending point
cpx2, *cpy2* – coordinates for the ending control point

- Example

```
size(200, 200);
background(255);
smooth();
stroke(0);
curve(40, 40, 80, 60, 100, 100, 60, 120);
curve(160, 40, 120, 60, 110, 100, 150, 130);
```



Method, Parameter, Argument

- The drawing commands are *methods*
- Here are the syntax of a method:

parameters

method name

x1, y1, x2, y2

```
line( 0, 0, 50, 50 );
```

arguments

parentheses contain parameters/arguments

Method, Parameter, Argument

- Methods are reusable commands
  - Like a little machine that does work for you
  - Let you reuse code without typing it over and over

- *Parameters* are placeholders for arguments
- *Arguments* tell the method (when it's called) precisely what to do

- We'll see later that you can define your own methods!

Variables

- A variable is a named box for storing a value
- You can put values in a variable by using the assignment operator (aka "=")

  e.g. `x = 1;`
- To use the value stored in a variable, just use the variable's name

  e.g. `line( x, 0, 50, 50);`

Variables have a type

- You must tell Processing (Java) what kind of values can go in the box

- You do this by giving a variable a *type*

```
int x;  // variable x can hold integers (int)
int y;  // variable y can hold integers (int)

x = 20;  // store 20 in x
y = 30;  // store 30 in y
point(x, y);  // use the values of x and y to draw a point
```

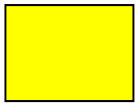
Creating an int variable

Code

```
// Single int
int anInt;
```

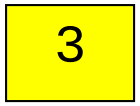
Effect

Name: **anInt**, Type: **int**



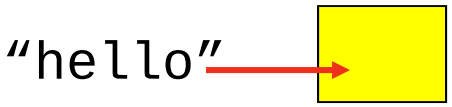
```
// Put a value in the int
anInt = 3;
```

Name: **anInt**, Type: **int**




```
// Type error!
anInt = "hello";
```

Name: **anInt**, Type: **int**



Can't shove "hello" into an int

Assigned values must match the type

```
int x;          // variable x can hold integers (int)
int y;          // variable y can hold integers (int)

x = 1.5;        // store 1.5 in x  causes an error!!!
y = 30;         // store 30 in y
point(x, y);    // use the values of x and y to draw a point
```

Why Types?

- Tells system (and you) what kind of values to expect
- System uses type to detect errors

```
int pi = 3.14 ; // error:3.14 not an int
```

- To walk around, using type casting:

```
int pi = (int) 3.14 ;
print(pi)→ 3
```

The “primitive” types

int – integers between -2,147,483,648 and 2,147,483,647

float – floating point numbers (e.g. 3.1415927, -2.34)

char – a single character (e.g. ‘c’)

byte – integers between -128 and 127

boolean – holds the values *true* or *false*

Can combine declaring and assigning

- Declaring a variable means telling Processing its type

  int x;

- Assigning a value to a variable means putting a value in the named box

  x = 1;

- You can declare and assign at the same time

  int x = 1;

- But only declare a variable once, otherwise you get an error

print and println

- When working with variables, it is often convenient to look at their values

- print() and println() print to the bottom processing pane
  - They do the same thing, except println starts a new line after printing

How to animate in Processing

- How can you get Processing to animate?

`setup()`

`void setup()` is a builtin Processing method **that you need to define** (i.e. plug in your code within its syntax{ })

`setup()` is called once when a sketch first starts executing

- Place any startup code in setup(), eg.
  - Setting the window size
  - Setting the background color, smooth
  - Initializing variables…

`draw()`

`draw()` is a builtin Processing method **that you need to define**

`draw()` is called repeatedly by the Processing system

- Put code in draw() when you need to constantly update the display (for example, animating an object)

Example of setup() and draw()

```
int x;
int y;

void setup() {
   size(400, 400);
   background(0);
   x = 0;
   y = height/2;
}

void draw() {
   background(0);
   ellipse(x, y, 20, 20);
   x = x + 1;
}
```

setup() and draw() are examples of *callbacks*

- A callback function is defined by the programmer
  - You usually don't call callback functions directly within your own code
  - The callback gets called in response to some internal events
  - setup() and draw() are predefined within Processing as to-be-called-if-defined

Controlling draw()

- *frameRate()* can be used to set the number of times per second that draw() is called
  - frameRate(30) says to call draw() 30 times a second (if the computer is capable)
- *delay()* delays execution for a certain number of milliseconds
  - delay(250) delays for 250 milliseconds (1/4 of a sec.)
  - You can use delay() or frameRate() to determine how fast you want draw() to be called – frameRate() is probably easier
- *noLoop()* tells the system to stop calling draw()
  - If you want to, for example, turn off animation
- *loop()* tells the system to resume calling draw() again
  - Use noLoop() and loop() together to turn repeated drawing on and off

Summary

- Multimedia as a Context
- Java and Processing
- Drawing primitives
- Method, parameter, argument
- Processing setup() & draw()
- Variables of Primitive Types
- Animation with setup() & draw()