Data Structures in Processing

The Object Class

- "Object" is the most general class
- Variables of type Object can hold any other type

```
Object myObj1 = new String("abc");
Object myObj2 = new PImage(100, 100);
Object myObj3 = 123;
```

- Variables of type Object don't know the type they hold, so the compiler can't check for legal operations.

The Object Class

- Constructors

```
Object o = new Object();
```

- Fields
- Methods

```
equals(Object o2)   // Tests for equality with Object o2
toString()          // Returns a String representation of Object
…
```

Type Casting

- We learned about type-conversion functions

```
int( … ), float( … ), boolean( … ), …
```

- Another way to convert from one type to another is called "type casting," which works by preceding an expressing with the target type in parentheses.

```
float f = 12.0;
int i = (int)f;  // Will not work without type cast

Object o = new PImage(100, 100);
PImage p = (PImage)o;
```

Built-in Collection Classes

- ArrayList
  - A built-in object that stores and manages an *arbitrary* number of data items of any type (Objects).
  - Objects in an ArrayList are access by **index** [0..size-1]

- HashMap
  - A built-in object that stores and manages an *arbitrary* number of data items of any type (Objects).
  - Objects in a HashMap are access by a **key**, which can be another Object, frequently a String.

ArrayList

- Constructors

```
ArrayList<Object> lst1 = new ArrayList();
ArrayList<Object> lst2 = new ArrayList(initialSize);
```

- Fields
- Methods

```
size()                          // Returns the num of items held.
add(Object o)                   // Appends o to end.
add(int idx, Object o)          // Inserts o at pos idx.
remove(int idx)                 // Removes item at pos idx.
get(int idx)                    // Gets items at idx. No removal.
set(int idx, Object o)          // Replaces item at idx with o.
clear()                         // Removes all items.
isEmpty()                       // true if empty.
```

ArrayList Example – Box Dropper

```
// Box Dropper
ArrayList<Box> boxes = new ArrayList();

void setup() { size(500, 500); }

void draw() {
  background(0);

  for (int i = boxes.size()-1; i>=0; i--) {
    //boxes.get(i).draw();      // Fails. Why?
    Box b = boxes.get(i);  // Type cast Object->Box
    b.y = b.y + b.v;           // Physics
    b.v = b.v + 0.02;
    b.draw();

    // Remove Box from ArrayList if below sketch
    if (b.y > height) {
      boxes.remove(i);
      println(boxes.size() + " boxes remaining");
    }
  }
}

void mousePressed() {
  Box b = new Box(mouseX, mouseY);
  boxes.add( b );
  println( boxes.size() + " boxes in ArrayList" );
}
```

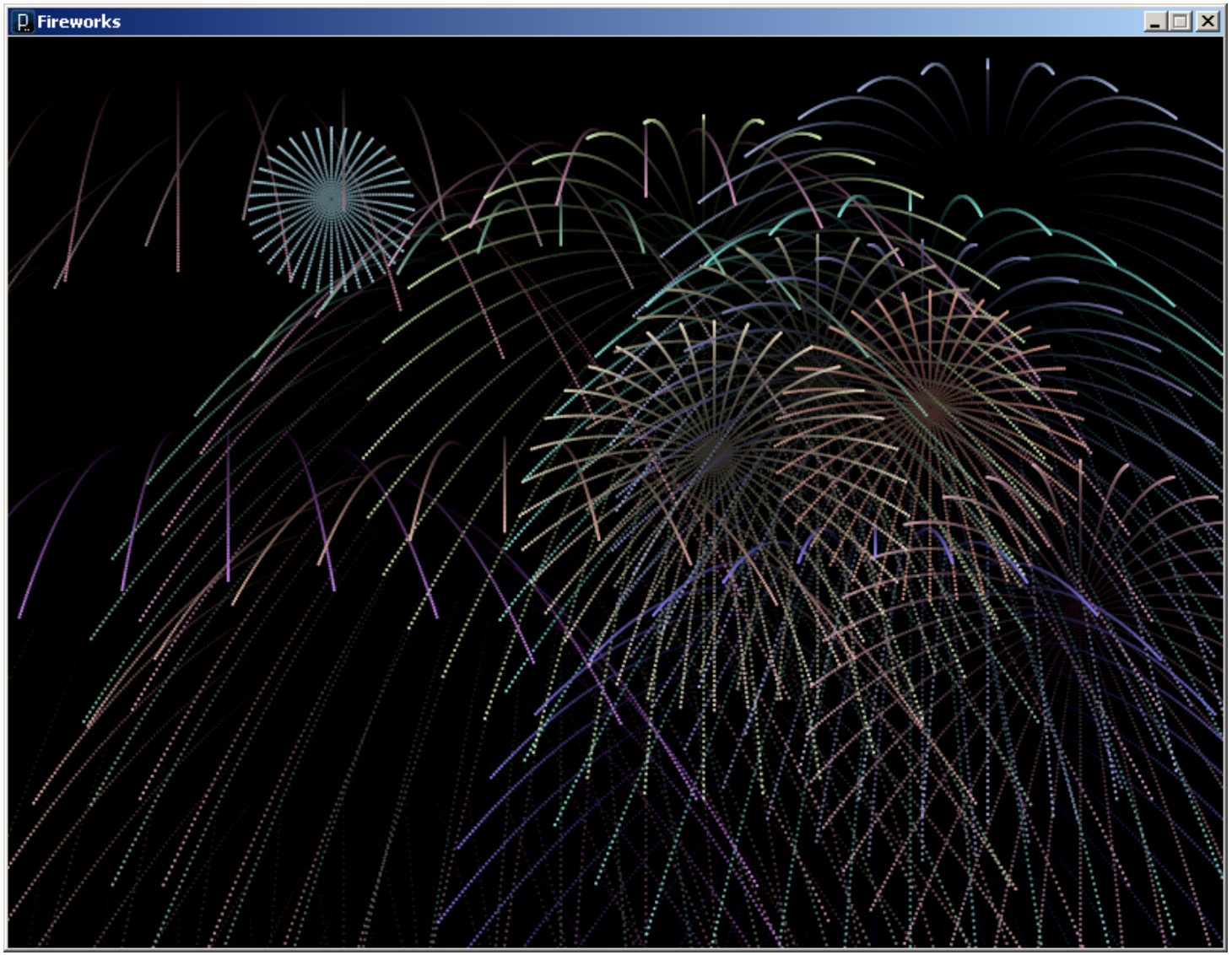
```
// A simple Box class
class Box {
  float x, y, v;

  Box(float tx, float ty) {
    x = tx;  // x position
    y = ty;  // y position
    v = 0.0; // y velocity
  }

  void draw() {
    fill(200);
    rect(x, y, 20, 20);
  }
}
```

- Why can we not call draw directly on item in ArrayList?
- Why do we loop over ArrayList backwards?

ArrayList Example - Fireworks



Fireworks Code

Modified from http://www.openprocessing.org/visuals/?visualID=30877

```
class Fire{
  float x;
  float y;
  float vx;
  float vy;

  color col;

  float lifetime = 100;

  Fire(float x, float y, float vx,
       float vy, color col){
    this.x = x;
    this.y = y;
    this.vx = vx;
    this.vy = vy;
    this.col = col;
  }

  void draw() {
    if(lifetime-50>0){
      noStroke();
      fill(col, lifetime-50);
      ellipse(x,y,4,4);
      lifetime -= 0.5;
    }
  }

  void update() {
    vy += G;
    x += vx;
    y += vy;
  }
}
```

Fireworks Code

```
ArrayList<Fire> fireworks = new
ArrayList();

final int FIRE_COUNT = 1000;
final float X = 200;
final float Y = 50;
final float G = 0.04;

void setup() {
   size(400,400);
}

void draw() {
  noStroke();
  background(0);

  for(Fire fire : fireworks){
    fire.update();
    fire.draw();
  }
}
```

```
void mousePressed() {
   fireworks.clear();

   color c = color(
               random(50,255),
               random(50,255),
               random(50,255));

   for(int i=0; i<FIRE_COUNT; i++){
    float r = random(0,TWO_PI);
    float R = random(0,2);

    fireworks.add(
        new Fire(mouseX,mouseY,
              R*sin(r),R*cos(r),c));
   }
}
```

HashMap

- Constructors

```
HashMap<Object,Object> map1 = new HashMap();
HashMap<Object,Object> map2 = new HashMap(initialCapacity);
```

- Fields
- Methods

```
size()                          // Returns num of items held.
put(Object key, Object o)       // Puts o in map at key
remove(Object key)              // Remove Object at key
get(Object key)                 // Get Object at key
containsKey(Object key)         // True if map contains key
containsValue(Object val)       // True if map contains val
clear()                         // Removes all items.
isEmpty()                       // true if empty.
```

HashMap Example – High Score

```
// HighScore
HashMap<String,Integer> scores =
   new HashMap();

void setup() {
  size(500, 500);

  // Init HashMap
  scores.put("Fred", 2);
  scores.put("Wilma", 4);
  scores.put("Barney", 10);
  scores.put("Betty", 5);
  scores.put("BamBam", 6);
  scores.put("Pebbles", 5);

  // Draw once
  noLoop();
  drawMap(scores);
}

void draw() { }
```

```
// Draw the HashMap to the sketch
void drawMap(HashMap hm) {
  background(0);
  fill(255);
  textSize(20);

  // Display all scores
  text( buildScore("Fred", scores), 100, 100);
  text( buildScore("Wilma", scores), 100, 150);
  text( buildScore("Barney", scores), 100, 200);
  text( buildScore("Betty", scores), 100, 250);
  text( buildScore("BamBam", scores), 100, 300);
  text( buildScore("Pebbles", scores), 100, 350);

  redraw();
}

// Build a return a String for displaying a Score
String buildScore(String name, HashMap hm) {
  String msg = name + ":" + hm.get(name).toString();
  return msg;
}
```

Sorting

- Selection Sort
  - Scan a list top to bottom and find the value that should come first
  - Swap that item with the top position
  - Repeat scan starting at next lowest item in the list
  - Works best when swapping is expensive

Selection Sort

```
// Selection Sort Example
ArrayList list = new ArrayList();
int start = 0;

void setup() {
  size(500, 500);

  // Fill the ArrayList
  list.add("Purin");
  list.add("Landry");
  list.add("Chococat");
  list.add("Pekkle");
  list.add("Cinnamoroll");

  noLoop(); // Draw once
  drawList(list);
}

void draw() { }

// Perform one pass of selection sort
void mousePressed() {
  selectOnce(list, start);
  if (start < list.size()-1) start++;
  //selectionSort(list);
}

// Perform a complete Selection Sort
void selectionSort(ArrayList al) {
  for (int i=0; i<al.size(); i++) {
    selectOnce(al, i);
  }
}
```

```
// Perform once pass of Selection Sort.
void selectOnce(ArrayList al, int i) {

  String bestVal = (String)al.get(i);
  int bestIdx = i;

  for (int j=i+1; j<al.size(); j++) {
    String s1 = (String)al.get(j);
    if (s1.compareTo(bestVal) < 1) {
      bestVal = (String)al.get(j);
      bestIdx = j;
    }
  }

  // Swap best with top position
  al.set(bestIdx, (String)al.get(i));
  al.set(i, bestVal);

  drawList(al); // Redraw list
  delay(1000);
}

// Draw the ArrayList to the sketch
void drawList(ArrayList al) {
  background(0);
  fill(255);
  textSize(20);

  int y=100;
  for (int i=0; i<al.size(); i++) {
    String s = (String)al.get(i);
    text(s, 100, y);
    y=y+50;
  }
  redraw();
}
```

Sorting

- Bubblesort
  - Scan through a list from top to bottom
  - Compare successive adjacent pairs of items
  - If two items are out of order, swap them
  - Repeat scan until no changes are made (completely ordered)
  - Works best when there are few items out of order

Bubble Sort

```
// Bubblesort Example
ArrayList list = new ArrayList();

void setup() {
  size(500, 500);

  // Fill the ArrayList
  list.add("Purin");
  list.add("Landry");
  list.add("Chococat");
  list.add("Pekkle");
  list.add("Cinnamoroll");

  // Draw once
  noLoop();
  drawList(list);
}

void draw() { }

// On mousePressed, bubble once
void mousePressed() {
  bubbleOnce(list);
  //bubbleSort(list);
}

// Perform a complete Bubblesort
void bubbleSort(ArrayList al) {
  while ( true ) {
    if (bubbleOnce(al) == false) break;
  }
}
```

```
// Perform once pass of Bubblesort.
// Return true if any changes.
boolean bubbleOnce(ArrayList al) {
  boolean changed = false;

  // Loop over all pairs
  for (int i=0; i<al.size()-1; i++) {
    String s1 = (String)al.get(i);
    String s2 = (String)al.get(i+1);

    // Swap if pair is not in order
    if (s1.compareTo(s2) > 0) {
      list.set(i, s2);
      list.set(i+1, s1);
      changed = true;

      drawList(al); // Redraw list if changed
      delay(1000);
    }
  }
  return changed;
}

// Draw the ArrayList to the sketch
void drawList(ArrayList al) {
  background(0);
  fill(255);
  textSize(20);

  int y=100;
  for (int i=0; i<al.size(); i++) {
    String s = (String)al.get(i);
    text(s, 100, y);
    y=y+50;
  }
  redraw();
}
```

Sorting Algorithm Animations

SHARE

Problem Size: 20 · 30 · 40 · 50 Magnification: 1x · 2x · 3x

Algorithm: Insertion · Selection · Bubble · Shell · Merge · Heap · Quick · Quick3

Initial Condition: Random · Nearly Sorted · Reversed · Few Unique

	Insertion	Selection	Bubble	Shell	Merge	Heap	Quick	Quick3
Random								
Nearly Sorted								
Reversed								
Few Unique								

http://www.sorting-algorithms.com/