Design Programming

DECO1012
Rob Saunders
Rob Saunders

web: http://web.arch.usyd.edu.au/~rob

e-mail: rob@arch.usyd.edu.au

office: Room 274, Wilkinson Building
Super Classes!
Composite Objects

- An object can contain other objects, just like any other type of data, as fields
- This process of building an object by combining other objects is called composition
class EggRing {
    Egg ovoid;
    Ring circle = new Ring();

    EggRing(int x, int y, float t, float sp) { 
        ovoid = new Egg(x, y, t, sp);
        circle.start(x, y - sp / 2);
    }

    void transmit() { 
        ovoid.wobble();
        ovoid.display();
        circle.grow();
        circle.display();
        if (circle.on == false) {
            circle.on = true;
        }
    }
}
Inheritance

- A class can be defined using another class as a foundation.
  - In object-oriented programming terminology, a subclass inherits fields and methods from a superclass.

- A subclass extends a superclass.
  - When one class extends another, all of the methods and fields from the superclass are automatically included in the subclass.
Extending a Superclass

- New fields and methods can be added to the subclass to build on the data and behavior of its superclass.
  - A method in a subclass can override those in a superclass if it has the same name and the same parameters.
  - To call a method in the superclass from the subclass, use the `super` keyword and the dot syntax to identify the method.
  - To call a superclass constructor use `super()`
class Spin {
    float x, y, speed;
    float angle = 0.0;

    Spin(float xpos, float ypos, float s) {
        x = xpos;
        y = ypos;
        speed = s;
    }

    void update() {
        angle += speed;
    }
}
class SpinArm extends Spin {
    SpinArm(float x, float y, float s) {
        super(x, y, s);
    }

    void display() {
        strokeWeight(1);
        stroke(0);
        pushMatrix();
        translate(x, y);
        angle += speed;
        rotate(angle);
        line(0, 0, 100, 0);
        popMatrix();
    }
}
class SpinSpots extends Spin {
  float dim;

  SpinSpots(float x, float y, float s, float d) {
    super(x, y, s);
    dim = d;
  }

  void display() {
    noStroke();
    pushMatrix();
    translate(x, y);
    angle += speed;
    rotate(angle);
    ellipse(-dim / 2, 0, dim, dim);
    ellipse(dim / 2, 0, dim, dim);
    popMatrix();
  }
}
SpinSpots spots;
SpinArm arm;

void setup() {
    size(100, 100);
    smooth();
    arm = new SpinArm(width / 2, height / 2, 0.01);
    spots = new SpinSpots(width / 2, height / 2, -0.02, 33.0);
}

void draw() {
    background(204);
    arm.update();
    arm.display();
    spots.update();
    spots.display();
}
BYOGUI
Build Your Own Graphical User Interface
Graphical User Interfaces

- As sketches become more complex, we often need a graphical user interface.
- For simple sketches with only a few parameters, keyboard inputs on their own work fine, but for controlling every aspect of a complex sketch it becomes increasingly difficult to remember them.
- Object-oriented programming provides a convenient way to build your own GUI.
Sensitive Shapes

- The first step in building a graphical user interface is to develop some simple shape classes that can react to the mouse and/or keyboard.
- We’re only going to look at simple shapes and mouse input in this lecture.
- The classes we’re going to define are called `OverCircle` and `OverRect`, these classes change their state (gray) when the mouse is hovering over them.
class OverCircle {
    int x, y;            // The x- and y-coordinates
    int diameter;        // Diameter of the circle
    int gray;            // Gray value

    OverCircle(int xp, int yp, int d) {
        x = xp;
        y = yp;
        diameter = d;
        gray = 0;
    }

    void update() {
        if (dist(mouseX, mouseY, x, y) < diameter / 2) {
            gray = 255;
        } else {
            gray = 0;
        }
    }

    void display() {
        fill(gray);
        ellipse(x, y, diameter, diameter);
    }
}
class OverRect {
    int x, y; // The x- and y-coordinates
    int size; // Dimension (width and height) of the rectangle
    int gray; // Gray value

    OverRect(int xp, int yp, int s) {
        x = xp;
        y = yp;
        size = s;
        gray = 0;
    }

    void update() {
        if ((mouseX > x) && (mouseX < x + size) &&
            (mouseY > y) && (mouseY < y + size)) {
            gray = 255;
        } else {
            gray = 0;
        }
    }

    void display() {
        fill(gray);
        rect(x, y, size, size);
    }
}
// Requires the OverRect and OverCircle classes
OverRect r = new OverRect(9, 30, 36);
OverCircle c = new OverCircle(72, 48, 40);

void setup() {
  size(100, 100);
  noStroke();
  smooth();
}

void draw() {
  background(204);
  r.update();
  r.display();
  c.update();
  c.display();
}
A Button Class

- To define a Button class we’ll extend the OverRect class to respond to mouse presses and releases and maintain a pressed field to represent its state.
  - The update() method records whether the mouse is over the button.
  - The press() method checks that the mouse is over the button and updates its state.
  - The release() method checks that the mouse is over the button and updates its state.
class Button {
    int x, y; // The x- and y-coordinates
    int size; // Dimension (width and height)
    color baseGray; // Default gray value
    color overGray; // Value when mouse is over the button
    color pressGray; // Value when mouse is over and pressed
    boolean over = false; // True when the mouse is over
    boolean pressed = false; // True when the mouse is over and pressed

    Button(int xp, int yp, int s, color b, color o, color p) {
        x = xp;
        y = yp;
        size = s;
        baseGray = b;
        overGray = o;
        pressGray = p;
    }

    // Updates the over field every frame
    void update() {
        if ((mouseX >= x) && (mouseX <= x + size) &&
            (mouseY >= y) && (mouseY <= y + size)) {
            over = true;
        } else {
            over = false;
        }
    }
}
void press() {
    if (over == true) {
        pressed = true;
    }
}

void release() {
    pressed = false; // Set to false when the mouse is released
}

void display() {
    if (pressed == true) {
        fill(pressGray);
    } else if (over == true) {
        fill(overGray);
    } else {
        fill(baseGray);
    }
    stroke(255);
    rect(x, y, size, size);
}
Using the **Button** Class

- To use the Button class we need to call the `update()`, `press()` and `release()` methods
  - The `update()` method should be called on every frame from within `draw()`
  - The `press()` method should be called on every mouse button press from within `mousePressed()`
  - The `release()` method should be called on every mouse button release from within `mouseReleased()`
Button button;

void setup() {
    size(100, 100);
    button = new Button(25, 25, 50, color(204), color(255), color(0));
}

void draw() {
    background(204);
    stroke(255);
    button.update();
    button.display();
}

void mousePressed() {
    button.press();
}

void mouseReleased() {
    button.release();
}
A Check Box Class

- A checkbox is similar to a button and we can define a similar class
- The difference between a checkbox and a button is that when the mouse is released a checkbox stays pressed
- The **Check** class implements this behaviour, notice that it only needs to define the `press()` method because its state only changes when it is pressed
class Check {
    int x, y; // The x- and y-coordinates
    int size; // Dimension (width and height)
    color baseGray; // Default gray value
    boolean over = false; // True when the mouse is over
    boolean checked = false; // True when the check box is selected

    Check(int xp, int yp, int s, color b) {
        x = xp;
        y = yp;
        size = s;
        baseGray = b;
    }

    // Updates the over field every frame
    void update() {
        if ((mouseX >= x) && (mouseX <= x + size) &&
            (mouseY >= y) && (mouseY <= y + size)) {
            over = true;
        } else {
            over = false;
        }
    }
}
// Updates the boolean variable checked
void press() {
  if (over == true) {
    checked = !checked; // Toggle the check box on and off
  }
}

// Draws the box and an X inside if the checked variable is true
void display() {
  stroke(255);
  fill(baseGray);
  rect(x, y, size, size);
  if (checked == true) {
    line(x, y, x + size, y + size);
    line(x + size, y, x, y + size);
  }
}

Check check;

void setup() {
  size(100, 100);
  // Inputs: x, y, size, fill color
  check = new Check(25, 25, 50, color(0));
}

void draw() {
  background(204);
  check.update();
  check.display();
}

void mousePressed() {
  check.press();
}
/Requires the Check class

```java
int numChecks = 25;
Check[] checks = new Check[numChecks];

void setup() {
    size(100, 100);
    int x = 14;
    int y = 14;
    for (int i = 0; i < numChecks; i++) {
        checks[i] = new Check(x, y, 12, color(0));
        x += 15;
        if (x > 80) { x = 14; y += 15; }
    }
}

void draw() {
    background(0);
    for (int i = 0; i < numChecks; i++) {
        checks[i].update();
        checks[i].display();
    }
}

void mousePressed() {
    for (int i = 0; i < numChecks; i++) {
        checks[i].press();
    }
}
```
A Radio Button Class

- Another common type of button used in user interfaces is a radio button
  - A radio button is much like a check box, it changes state when pressed
  - Unlike a check box only one radio button (in a group) can be “on” at a time
class Radio {
    int x, y; // The x- and y-coordinates of the rect
    int size, dotSize; // Dimension of circle, inner circle
    color baseGray, dotGray; // Circle gray value, inner gray value
    boolean over = false; // True when the mouse is over
    boolean checked = false; // True when the button is selected
    int me; // ID number for this Radio object
    Radio[] others; // Array of all other Radio objects

    Radio(int xp, int yp, int s, color b, color d, int m, Radio[] o) {
        x = xp;
        y = yp;
        size = s;
        dotSize = size - size / 3;
        baseGray = b;
        dotGray = d;
        others = o;
        me = m;
    }

    // Updates the over field every frame
    void update() {
        if (dist(mouseX, mouseY, x, y) < diameter / 2) {
            over = true;
            gray = 255;
        } else {
            over = false;
            gray = 0;
        }
    }
}
// Updates the boolean value press, returns true or false
void press(float mx, float my) {
    if (over == true) {
        checked = true;
        for (int i = 0; i < others.length; i++) {
            if (i != me) others[i].checked = false;
        }
        return true;
    } else {
        return false;
    }
}

// Draws the element to the display window
void display() {
    noStroke();
    fill(baseGray);
    ellipse(x, y, size, size);
    if (checked == true) {
        fill(dotGray);
        ellipse(x, y, dotSize, dotSize);
    }
}
Radio[] buttons = new Radio[2];

void setup() {
    size(100, 100);
    smooth();
    // Inputs: x, y, size, base color, fill color, id number, array of others
    buttons[0] = new Radio(33, 50, 30, color(255), color(0), 0, buttons);
    buttons[1] = new Radio(66, 50, 30, color(255), color(0), 1, buttons);
}

void draw() {
    background(204);
    buttons[0].update();
    buttons[0].display();
    buttons[1].update();
    buttons[1].display();
}

void mousePressed() {
    buttons[0].press();
    buttons[1].press();
}
A Scrollbar Class

- A scrollbar needs to react to mouse being pressed and released over a handle that can be moved along a (constrained) path
class Scrollbar {
    int x, y; // The x- and y-coordinates
    float sw, sh; // Width and height of scrollbar
    float pos; // Position of thumb
    float posMin, posMax; // Max and min values of thumb
    boolean rollover; // True when the mouse is over
    boolean locked; // True when its the active scrollbar
    float minVal, maxVal; // Min and max values for the thumb

    Scrollbar(int xp, int yp, int w, int h, float miv, float mav) {
        x = xp; y = yp;
        sw = w; sh = h;
        minVal = miv; maxVal = mav;
        pos = x + sw / 2 - sh / 2;
        posMin = x; posMax = x + sw - sh;
    }

    // Draws the scrollbar to the screen
    void display() {
        fill(255);
        rect(x, y, sw, sh);
        if ((rollover == true) || (locked == true)) {
            fill(0);
        } else {
            fill(102);
        }
        rect(pos, y, sh, sh);
    }
}
// Updates the over boolean and the position of the thumb
void update() {
    if (over() == true) {
        rollover = true;
    } else {
        rollover = false;
    }
    if (locked == true) {
        pos = constrain(mouseX - sh / 2, posMin, posMax);
    }
}

// Returns true if the cursor is over the scrollbar
boolean over() {
    if ((mouseX > x) && (mouseX < x + sw) &&
        (mouseY > y) && (mouseY < y + sh)) {
        return true;
    } else {
        return false;
    }
}
void press() {
    if (rollover == true) {
        locked = true;
    } else {
        locked = false;
    }
}

void release() {
    locked = false;
}

float getPos() {
    float scalar = sw / (sw - sh);
    float ratio = (pos - x) * scalar;
    float offset = minVal + (ratio / sw * (maxVal - minVal));
    return offset;
}
Extending the **Button** Class

- The following DragButton class extends the Button class, allowing a button to be repositioned on the screen
  - It extends the behaviour of the `press()` method and adding support for a `drag()` method
class DragButton extends Button {
    int xoff, yoff;

    DragButton(int x, int y, int s, color bv, color ov, color pv) {
        super(x, y, s, bv, ov, pv);
    }

    void press(int mx, int my) {
        super.press();
        xoff = mx - x;
        yoff = my - y;
    }

    void drag(int mx, int my) {
        if (press == true) {
            x = mx - xoff;
            y = my - yoff;
        }
    }
}
DragButton icon;

void setup() {
    size(100, 100);
    smooth();
    color gray = color(204);
    color white = color(255);
    color black = color(0);
    icon = new DragButton(21, 42, 50, gray, white, black);
}

void draw() {
    background(204);
    icon.update(mouseX, mouseY);
    icon.display();
}

void mousePressed() {
    icon.press(mouseX, mouseY);
}

void mouseReleased() {
    icon.release();
}

void mouseDragged() {
    icon.drag(mouseX, mouseY);
}
class DragImage extends DragButton {
    PImage img;

    DragImage(int x, int y, int d, String s) {
        super(x, y, d, color(204), color(255), color(0));
        img = loadImage(s);
    }

    // Override the display() from Button
    void display() {
        if (press == true) {
            stroke(pressGray);
        } else if (over == true) {
            stroke(overGray);
        } else {
            stroke(baseGray);
        }
        noFill();
        rect(x - 1, y - 1, size + 1, size + 1);
        image(img, x, y, size, size);
    }
}
Lab Exercises
Lab Exercises

- Create a `CircularButton` class.
  - Look at the `OverCircle` and `Radio` classes.

- Extend the `Scrollbar` class to have arrow buttons on the left and right that move the thumb one step each time an arrow is pressed.
  - You might want to try doing this by creating a composite `ExtendedScrollbar` class that has a standard `Scrollbar` and two `Buttons` as fields.