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
Object Orientation
Object-Oriented Programming

- In our sketches so far, functions often work together on a set of related variables.
  - Object-oriented programming was developed to make this process more explicit.

- Object-oriented programming uses objects and classes as building blocks.
  - A class defines a group of methods (functions) and fields (variables). An object is a single instance of a class.
Object Orientation

› It’s possible to make an analogy between software objects and real-world artefacts.

<table>
<thead>
<tr>
<th>Name</th>
<th>Apple</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fields</td>
<td>colour, weight</td>
</tr>
<tr>
<td>Methods</td>
<td>grow(), fall(), rot()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Radio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fields</td>
<td>frequency, volume</td>
</tr>
<tr>
<td>Methods</td>
<td>turnOn(), tune(), setVolume()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fields</td>
<td>make, model, colour, year</td>
</tr>
<tr>
<td>Methods</td>
<td>accelerate(), brake(), turn()</td>
</tr>
</tbody>
</table>
The Apple Class

- The methods of the Apple class would affect the fields of an Apple object.
  - The `grow()` method could increase the weight field of the apple based on the environment.
  - The `fall()` method could check the weight and cause the apple to fall to the ground when the weight goes above a threshold.
  - The `rot()` method could then take over, beginning to decrease the value of the weight field and change the colour field.
Encapsulation

- Notice how the methods provide a simple interface to manipulating the fields.
- This allows programmers to hide a lot of the complexity of their objects.
- Try to imagine how complex the PImage and PFont classes must be... and yet they provide relatively simple interfaces using methods.
- Hiding details of the implementation is called encapsulation and is one of the big advantages of using objects.
Apple Objects

- An instance of a class is an object and can be assigned to a variable.
- If more than one object is created from a class, each must have a unique name.
- For example, if two objects named `fuji` and `golden` are created using the Apple class, each can have its own values for its fields:

<table>
<thead>
<tr>
<th>Name</th>
<th>Fields</th>
<th>Name</th>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>fuji</td>
<td>colour: red</td>
<td>golden</td>
<td>colour: yellow</td>
</tr>
<tr>
<td></td>
<td>weight: 0.175</td>
<td></td>
<td>weight: 0.2</td>
</tr>
</tbody>
</table>

Representations of Objects

Tables:

<table>
<thead>
<tr>
<th>Apple</th>
<th>fujii</th>
<th>golden</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>red</td>
<td>yellow</td>
</tr>
<tr>
<td>weight</td>
<td>6.2</td>
<td>8.4</td>
</tr>
<tr>
<td>grow()</td>
<td>grow()</td>
<td>grow()</td>
</tr>
<tr>
<td>fall()</td>
<td>fall()</td>
<td>fall()</td>
</tr>
<tr>
<td>rot()</td>
<td>rot()</td>
<td>rot()</td>
</tr>
</tbody>
</table>

Diagrams:

Apple class

fujii object

golden object
The fields and methods of an object are accessed with the dot operator.

- The syntax `fuji.colour` accesses the value of the `colour` field inside the `fuji` object.
- To run the `grow()` method inside the `golden` object, the syntax `golden.grow()` is used.
Using Objects
A Simple Example

int x = 33;
int y = 50;
int diameter = 30;

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

void draw() {
    background(0);
    ellipse(x, y, diameter, diameter);
}
The first step to defining an object might be to group together the variables that belong together as fields in a class...

class Spot {
    float x, y;   // The x- and y-coordinate
    float diameter;  // Diameter of the circle
}

Grouping Variable Together
Spot sp; // Declare the object

void setup() {
    size(100, 100);
    smooth();
    noStroke();
    sp = new Spot(); // Construct the object
    sp.x = 33; // Assign 33 to the x field
    sp.y = 50; // Assign 50 to the y field
    sp.diameter = 30; // Assign 30 to the diameter field
}

void draw() {
    background(0);
    ellipse(sp.x, sp.y, sp.diameter, sp.diameter);
}
Defining Methods

- The next step to defining an object might be to create methods to take over from functions that use an object’s fields...

```java
class Spot {
    float x, y; // The x- and y-coordinate
    float diameter; // Diameter of the circle

    void display() {
        ellipse(x, y, diameter, diameter);
    }
}
```
Spot sp;  // Declare the object

void setup() {
  size(100, 100);
  smooth();
  noStroke();
  sp = new Spot();  // Construct the object
  sp.x = 33;
  sp.y = 50;
  sp.diameter = 30;
}

void draw() {
  background(0);
  sp.display();
}
Defining a Constructor

- Constructors are special types of methods that create new objects that can take parameters for initialising fields etc.

```java
class Spot {
    ...

    Spot(float xpos, float ypos, float dia) {
        x = xpos;    // Assign xpos to x
        y = ypos;    // Assign ypos to y
        diameter = dia; // Assign dia to diameter
    }

    ...
}
```
class Spot {
    float x, y;       // The x- and y-coordinate
    float diameter;   // Diameter of the circle

    Spot(float xpos, float ypos, float dia) {
        x = xpos;       // Assign xpos to x
        y = ypos;       // Assign ypos to y
        diameter = dia; // Assign dia to diameter
    }

    void display() {
        ellipse(x, y, diameter, diameter);
    }
}
Spot sp; // Declare the object

void setup() {
  size(100, 100);
  smooth();
  noStroke();
  sp = new Spot(33, 50, 30); // Construct the object
}

void draw() {
  background(0);
  sp.display();
}
Extending the Spot Class

- The Spot class can now be extended to support more behaviours by:
  - Adding `speed` and `direction` fields to maintain information about movement
  - Adding a `move()` method to update the position of a Spot object when called
```java
class Spot {
    float x, y; // X-coordinate, y-coordinate
    float diameter; // Diameter of the circle
    float speed; // Distance moved each frame
    int direction = 1; // Direction of motion (1 is down, -1 is up)

    Spot(float xpos, float ypos, float dia, float sp) {
        x = xpos; y = ypos;
        diameter = dia;
        speed = sp;
    }

    void move() {
        y += (speed * direction);
        if ((y > (height - diameter / 2)) || (y < diameter / 2)) {
            direction *= -1; // Bounce of the window edges
        }
    }

    void display() {
        ellipse(x, y, diameter, diameter);
    }
}
```
Spot sp; // Declare the object

void setup() {
    size(100, 100);
    smooth();
    noStroke();
    sp = new Spot(33, 50, 30, 1.5);
}

void draw() {
    fill(0, 15);
    rect(0, 0, width, height);
    fill(255);
    sp.move();
    sp.display();
}

// Insert Spot class here...
Spot sp1, sp2, sp3; // Declare the objects

void setup() {
    size(100, 100);
    smooth();
    noStroke();
    sp1 = new Spot(20, 50, 40, 0.5);
    sp2 = new Spot(50, 50, 10, 2.0);
    sp3 = new Spot(80, 50, 30, 1.5);
}

void draw() {
    fill(0, 15);
    rect(0, 0, width, height);
    fill(255);
    sp1.move();
    sp2.move();
    sp3.move();
    sp1.display();
    sp2.display();
    sp3.display();
}
Arrays of Objects

- Working with arrays of objects is similar to working with arrays of other data types.
  - Because each array element is an object, each element of the array must be created before it can be accessed. The steps for working with an array of objects are:
    1. Declare the array
    2. Create the array
    3. Create each object in the array
int numSpots = 6;
Spot[] spots = new Spot[numSpots]; // Declare and create the array

void setup() {
  size(100, 100);
  smooth();
  noStroke();
  for (int i = 0; i < spots.length; i++) {
    float x = 10 + i * 16;
    float rate = 0.5 + i * 0.05;
    spots[i] = new Spot(x, 50, 16, rate); // Create each object
  }
}

void draw() {
  fill(0, 12);
  rect(0, 0, width, height);
  fill(255);
  for (int i = 0; i < spots.length; i++) {
    spots[i].move(); // Move each object
    spots[i].display(); // Display each object
  }
}
The Ring Class

- The Ring class presents another example of working with arrays and objects.
- This class defines a circle that can be turned on, at which point it expands to a width of 400 and then stops displaying to the screen by turning itself off.
class Ring {
    float x, y;  // X-coordinate, y-coordinate
    float diameter;  // Diameter of the ring
    boolean on = false;  // Turns the display on and off

    void start(float xpos, float ypos) {
        x = xpos; y = ypos; on = true; diameter = 1;
    }

    void grow() {
        if (on == true) {
            diameter += 0.5;
            if (diameter > 400) {
                on = false;
            }
        }
    }

    void display() {
        if (on == true) {
            noFill();
            strokeWeight(4);
            stroke(155, 153);
            ellipse(x, y, diameter, diameter);
        }
    }
}
Ring[] rings; // Declare the array
int numRings = 50;
int currentRing = 0;

void setup() {
    size(100, 100);
    smooth();
    rings = new Ring[numRings]; // Create the array
    for (int i = 0; i < numRings; i++) {
        rings[i] = new Ring(); // Create each object
    }
}

void draw() {
    background(0);
    for (int i = 0; i < numRings; i++) {
        rings[i].grow();
        rings[i].display();
    }
}

// Click to create a new Ring
void mousePressed() {
    rings[currentRing].start(mouseX, mouseY);
    currentRing++;
    if (currentRing >= numRings) currentRing = 0;
}
Lab Exercises
Lab Exercises

› Write your own unique Spot class that has a different behaviour than the one presented in the example.

› Design a kinetic composition with 90 of your Spots.

› Design a class that displays, animates, and defines the behaviour of an organism in relation to another object made from the same class.