01000011 01001111 01010011 01000011 01111011

COSC 123

Computer Creativity

Object Oriented Programming II

UBC Okanagan

Slides courtesy of Dr. Abdallah Mohamed.

Information about the Final Exam

- Check SSC for the official date and time of the Final Exam
- There will be some multiple choice questions, but the majority will be coding tasks
- The final exam will be:
  - **Cumulative**.
  - Live (2.5 hours), invigilated, but no proctoring.
  - Open book, open-notes, open-web but no cheating sites like Chegg/Course-Hero/Bartleby etc
  - IDEs are ok
  - On Canvas, hopefully using Gradescope

Announcements

- Next week (Week 11) will be the last week of labs in the course!

- Next Friday April 2nd is a holiday so lecture is cancelled, I have posted the lecture slides for reference and will post a lecture recording

- Office hours will resume as normal after Easter Monday.

- In Week 13 I will give you a preview of what's to come in future COSC courses and a guide for continuing in the COSC stream

Object Oriented Programming II

UBC Okanagan

Slides courtesy of Dr. Abdallah Mohamed.

Key Points

1) Using constructors to initialize objects as they are created.
2) Object references
3) Advanced: The *this* keyword
4) Advanced: Garbage collection and Object’s lifetime

Constructors

- A ***constructor*** is a special function that is **called when the object is first created to initialize** its attributes.
    - A constructor may have parameters like any other function.

```
Ball b = new Ball();
```

Calling the constructor

- Unlike other functions, a constructor:
    - 1) has a name that is the same as the class name.
    - 2) can not have a return type.
    - 3) is called only when we create the object using the new keyword.

- ***IF*** you do *not* supply a constructor for a class, Processing will use a ***default constructor*** which has no parameters which looks like this:

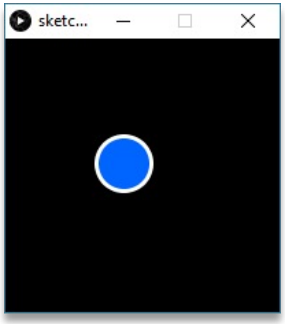
```
Ball(){ }; //attributes are set some default values
```

Example 1

Ball Class WITHOUT a Constructor

```
Ball ball;
void setup(){
  size(200,200);
  //create & initialize ball
  ball = new Ball();
  ball.x = 100;
  ball.y = 100;
  ball.speedX = 2;
  ball.speedY = 3;
  ball.r = 20;
}
void draw(){
  background(0);
  ball.moveBounce();
  ball.display();
}
```

```
class Ball {
 //attributes
 float x,y,r,speedX,speedY;
 //behavior
 void moveBounce() {
  x = x + speedX;
  y = y + speedY;
  if (x<r||x>width-r) speedX = -speedX;
  if (y<r||y>height-r)speedY = -speedY;
 }
 void display() {
  fill(0,100,255);
  stroke(255);
  strokeWeight(r/7);
  ellipse(x, y, 2*r, 2*r);
 }
}
```



Using Constructors

- Wouldn’t it be nice to initialize the attributes of an object as we create it?

```
// create a ball object
Ball ball = new Ball();
// initialize attributes
ball.x = 100;
ball.y = 100;
ball.speedX = 2;
ball.speedY = 3;
ball.r = 20;
```

```
// create a ball object and initialize it
Ball  ball=new Ball(100,100,2,3,20);
```

Example 1

Ball Class WITH a Constructor

```
Ball ball;
void setup(){
  size(200,200);
  //create & initialize a ball
  ball=new Ball(100,100,2,3,20);
}
void draw(){
  background(0);
  ball.moveBounce();
  ball.display();
}
```

```
class Ball {
 //attributes
 float x,y,r,speedX,speedY;
 //constructor
 Ball(float a, float b, float sx, float sy, float r1){
   x = a; y = b; r = r1;
   speedX = sx; speedY = sy;
 }
 //behavior
 void moveBounce() {
  x = x + speedX;
  y = y + speedY;
  if (x<r||x>width-r) speedX = -speedX;
  if (y<r||y>height-r)speedY = -speedY;
 }
 void display() {
  fill(0,100,255);
  stroke(255);
  strokeWeight(r/7);
  ellipse(x, y, 2*r, 2*r);
 }
}
```

More than One Constructor

- We can have more than one constructor in the same class as long as their parameters are different

- In this example, the `Ball` class defines two constructors:
  - a zero-argument constructor that sets the attributes to some default values.
  - A five-argument constructor that sets the attributes to given values.

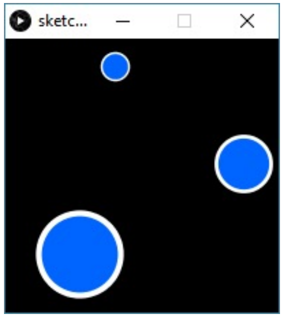
```
class Ball {
 //attributes
 float x,y,r,speedX,speedY;
 //constructors
 Ball(){
   x = 50; y = 50; r = 20;
   speedX=2; speedY=2;
 }
 Ball(float x1, float y1, float sx, float sy, float r1){
   x = x1; y = y1; r = r1;
   speedX=sx; speedY=sy;
 }
 //behavior
 void moveBounce() {
  x = x + speedX;  y = y + speedY;
  if (x<r||x>width-r) speedX = -speedX;
  if (y<r||y>height-r)speedY = -speedY;
 }
 void display() {
  fill(0,100,255);
  stroke(255); strokeWeight(r/7);
  ellipse(x, y, 2*r, 2*r);
 }
}
```

Example 2

SEVERAL Balls

```
Ball b1,b2,b3;
void setup(){
  size(200,200);
  //create & initialize a ball
  b1 = new Ball(100,100,2,3,20);
  b2 = new Ball();
  b3 = new Ball(80,70,2,-3,30);
}
void draw(){
  background(0);
  b1.moveBounce(); b1.display();
  b2.moveBounce(); b2.display();
  b3.moveBounce(); b3.display();
}
```

Q: How would the code look like if we don’t use constructors?



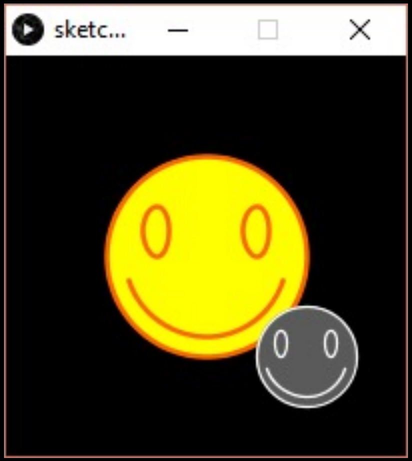
```
class Ball {
 //attributes
 float x,y,r,speedX,speedY;
 //constructor
 Ball(){
   x = 50; y = 50; r = 20;
   speedX=2; speedY=2;
 }
 Ball(float x1,float y1,float sx,float sy,float r1){
   x = x1; y = y1; r = r1;
   speedX=sx; speedY=sy;
 }
 //behavior
 void moveBounce() {
  x = x + speedX;
  y = y + speedY;
  if (x<r||x>width-r) speedX = -speedX;
  if (y<r||y>height-r)speedY = -speedY;
 }
 void display() {
  fill(0,100,255);
  stroke(255);
  strokeWeight(r/7);
  ellipse(x, y, 2*r, 2*r);
 }
}
```

Exercise 1

Add Constructors to HappyFace Class

[+2] 1) Modify your HappyFace class from previous unit so that it has two constructors:

- A zero-arg constructor that sets the radius to 50, the (x,y) position to (radius,radius), speedX and speedY to 0, fill color to yellow, outline color to orange.
- A seven-arg constructor that sets the attributes to given values.

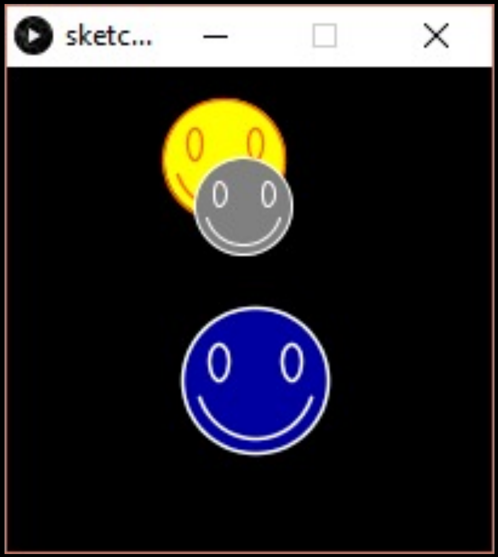


Exercise 1, cont'd

Bouncing Happy-Faces

[+2] 2) Create three bouncing happy-faces with different positions, size and speed, and then move, bounce, and display them in the draw() function.

- Notice how easy it is to create many objects now of the same class and use them in your program.



```
HappyFace f1, f2, f3;

void setup(){
  size(200,200);
  f1 = new HappyFace(...);
  //do the same for f2,f3
}

void draw(){
  background(0);
  f1.move;  f1.bounce(); f1.display();
  //do the same for f2,f3
}

class HappyFace{...}
```

Object References

Object References

- It is important to realize the difference between an object and an object reference.

- When you declare an object variable, you are actually declaring an object reference to that particular object type.
  - Until you create an object using `new`, there is no object in memory which is pointed to by the object reference.

- An object is the physical memory representation of the data.
  - An object has a location in memory and a type (class).
    - Each object has its own memory space and attribute values.

Changing Object References

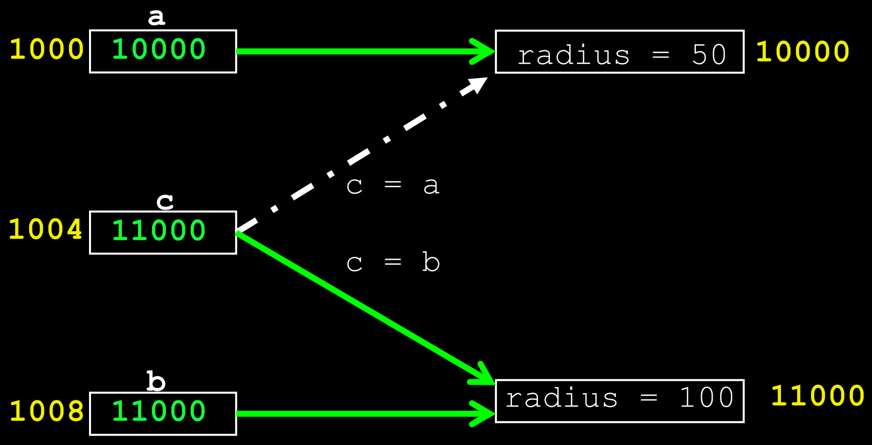
- Object references are pointers to objects in memory that can be assigned to the same value as another reference using '=' or assigned to `null` (which means they refer to nothing).

Example:

```
// “a” is an object reference to a Ball object
Ball a = new Ball(50);
// “b” is an object reference to a Ball object
Ball b = new Ball(100);
// “a” is an object reference (no objects created)
Ball c;

c = b;                 // c points to b
println(c.radius);     // 100
c = a;                 // c points to a
println(c.radius);     // 50
```

```
class Ball {
    float radius;
    Ball(float r){ radius = r; }
}
```



- Each ***object has its own space in memory*** AND each ***object reference*** also has its ***own memory space***.
- Object references point to objects and can be changed

`null` *Object References*

- Sometimes a programmer wants an object reference to point to ***nothing***. To make an object reference refer to nothing, you assign it a value of `null`.

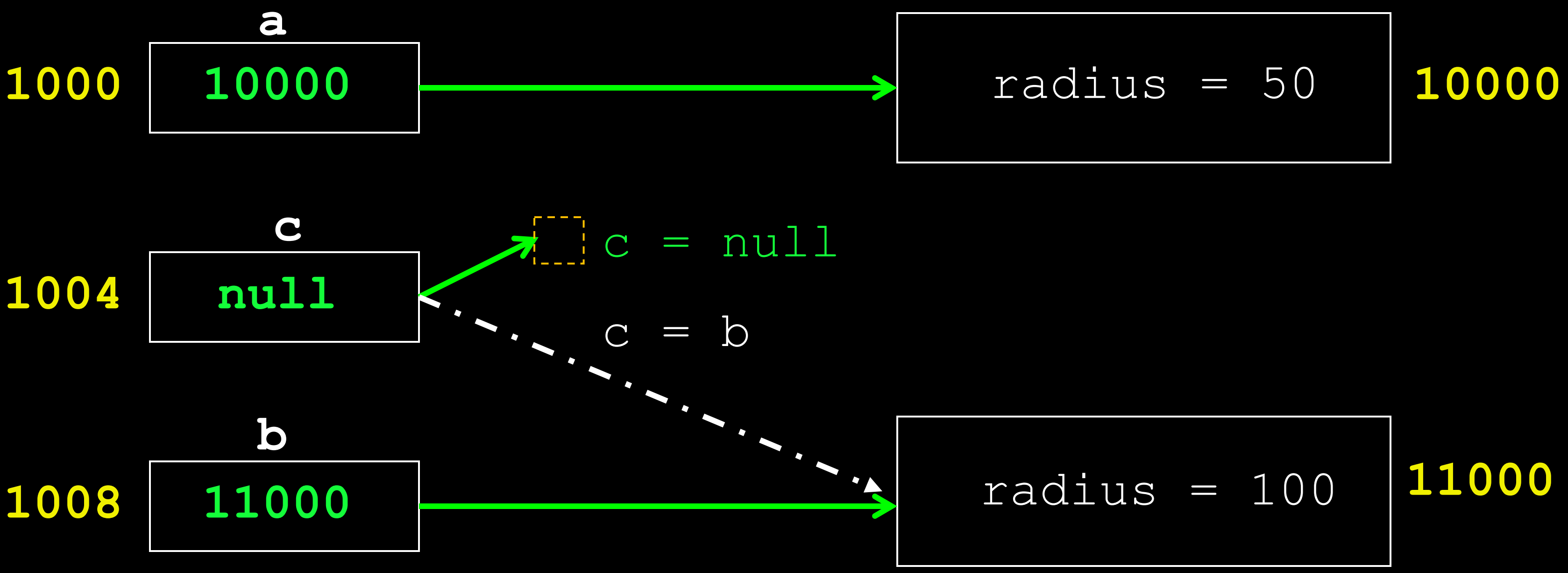
- Example:

```
Ball a = new Ball(50);
Ball b = new Ball(100);
Ball c;

c = b;               // c points to b
println(c.radius);   // 100
c = null;            // c now points to null
println(c.radius);   // Error!
```

`null` *Object References Example*

- A null reference effectively stores the address of **0**. Since this is not a valid memory address for the program, your program will generate a **run-time error** during execution.
    - The compiler does not check `null` references for you!
- Example:



Question

Objects and Object References

How many objects are created by this code?

```
Ball a, b, c;

a = new Ball();
c = a;
b = new Ball();
```

A. 1

B. 2

C. 3

D. 4

Question

Objects and Object References

What is the radius of the ball referenced by `d`?

```
Ball a, b, c, d;

a = new Ball(50);     // radius = 50
c = a;
b = new Ball(100);    // radius = 100
a = b;
d = c;
```

A. unknown

B. 50

C. 100

D. undefined

Advanced: using this

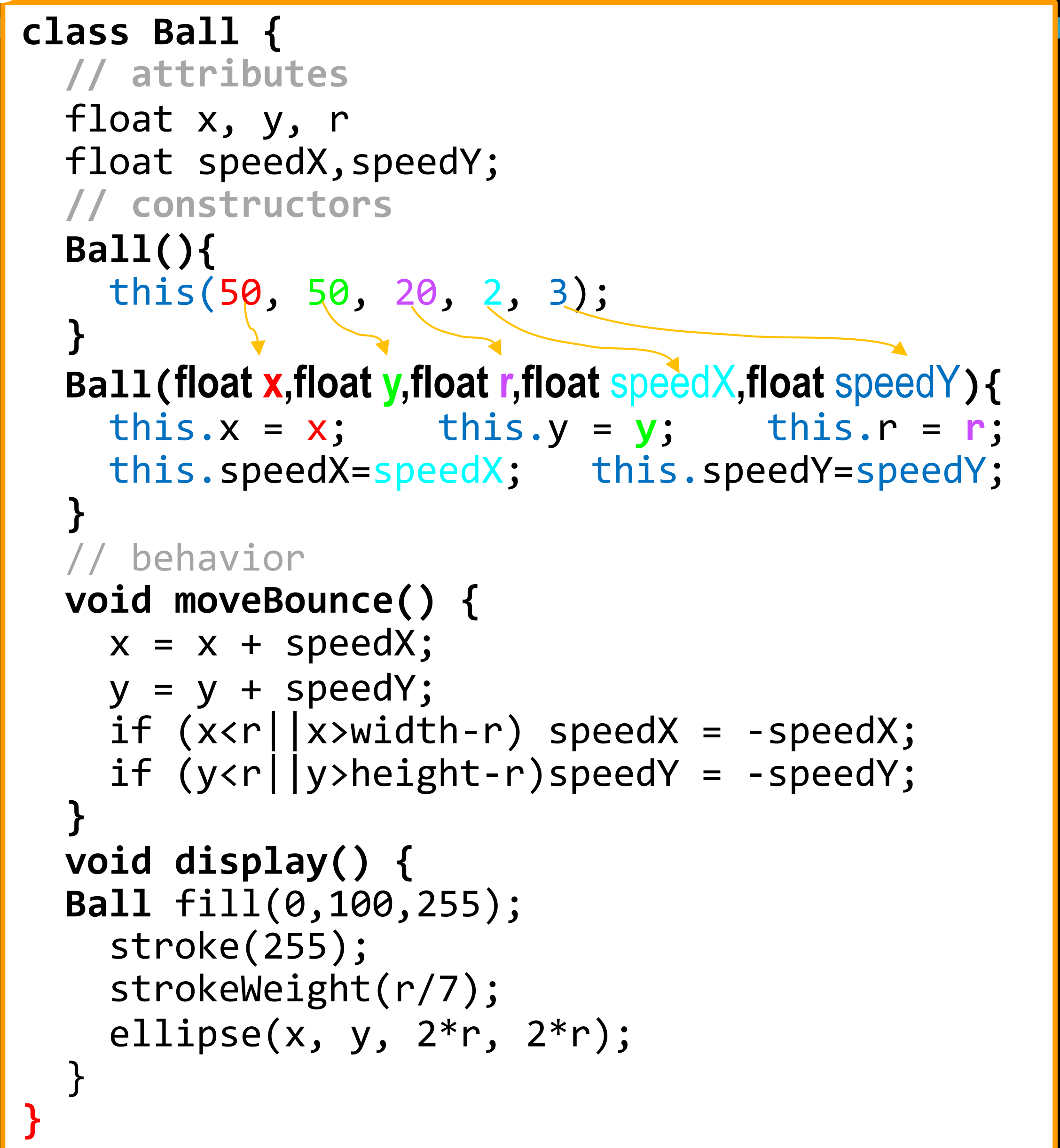
- When an object function is called, we tell Processing which object to use based on an object reference.
- This object reference is then accessible within an object functions as the **this** reference.

```
class Ball {
  // attributes
  float x, y, r
  float speedX,speedY;
  // constructors
  Ball(){
    this.x = 50; this.y = 50;     this.r = 20;
    this.speedX = 2;    this.speedY = 3;
  }
  Ball(float x,float y,float r,float speedX,float speedY){
    this.x = x;     this.y = y;     this.r = r;
    this.speedX=speedX;    this.speedY=speedY;
  }
  // behavior
  void moveBounce() {
    x = x + speedX;
    y = y + speedY;
    if (x<r||x>width-r) speedX = -speedX;
    if (y<r||y>height-r)speedY = -speedY;
  }
  void display() {
  Ball fill(0,100,255);
    stroke(255);
    strokeWeight(r/7);
    ellipse(x, y, 2*r, 2*r);
  }
}
```

must use this to distinguish between function parameters and object attributes as they have same name.

Advanced: using this

- the this reference can be used to call a constructor **from another constructor** in the **same class**



```
class Ball {
  // attributes
  float x, y, r
  float speedX,speedY;
  // constructors
  Ball(){
    this(50, 50, 20, 2, 3);
  }
  Ball(float x,float y,float r,float speedX,float speedY){
    this.x = x;    this.y = y;    this.r = r;
    this.speedX=speedX;   this.speedY=speedY;
  }
  // behavior
  void moveBounce() {
    x = x + speedX;
    y = y + speedY;
    if (x<r||x>width-r) speedX = -speedX;
    if (y<r||y>height-r)speedY = -speedY;
  }
  void display() {
  Ball fill(0,100,255);
    stroke(255);
    strokeWeight(r/7);
    ellipse(x, y, 2*r, 2*r);
  }
}
```

Advanced: Garbage Collection

- Have you ever wondered what happens to objects that you no longer need after you created them using **`new`**?
  - Unlike some other languages, a Java programmer is not responsible for deleting or destroying objects that you no longer use.
  - When an object has no valid references to it, Java may delete the object in memory in a process called ***garbage collection***.

Advanced: Object's Lifetime in Memory

- The lifetime of an object in memory:
  1) The object is created using `new` and a reference to its location in memory is created.
  2) The object may have multiple object references during the program execution.
  3) When all object reference variables go out of scope, the object has no more references and is marked for deletion.
  4) Java periodically scans memory and deletes objects.

Conclusion

- Key object-oriented terminology:
    - ***Object*** – an instance of a class.
    - ***Class*** – an object template with methods and properties.
    - ***Function (or Method)*** – a set of statements that performs an action.
    - ***Parameter*** – data passed into a method.
    - ***Properties*** – are attributes of objects.
- Object references point to objects in memory. Use `new` to create objects. Functions are called using an object reference.
- The scope and lifetime of a variable depends on its type (instance, static, local, parameter).