Game of Pong

Overview and starting development

Produced by: Dr. Siobhán Drohan
Mairead Meagher

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/

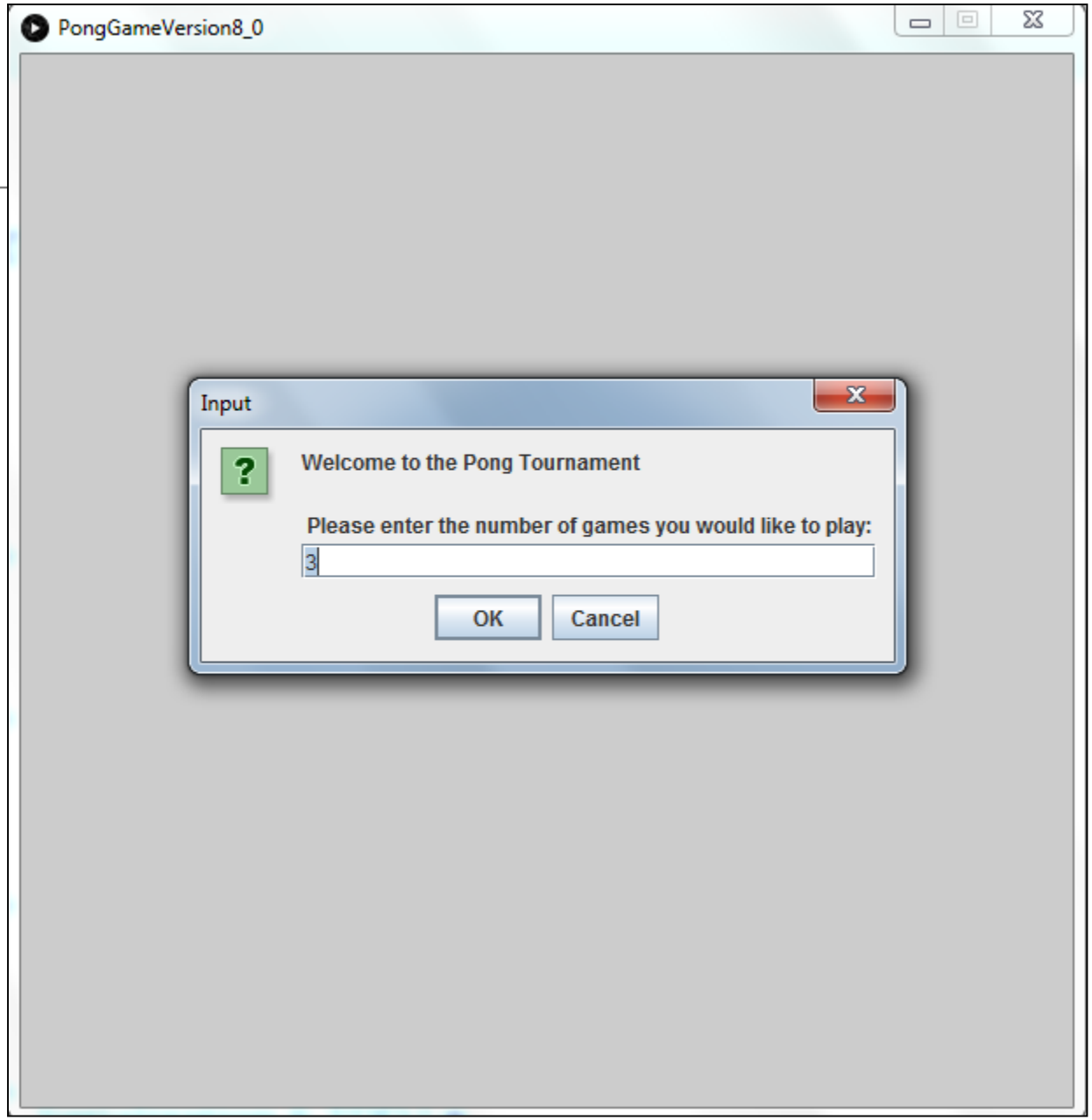
Topics list

- Overview of PongGameV8.0
- Developing:
  - PongGameV1.0 (Ball class)
  - PongGameV2.0 (Paddle class)
  - PongGameV3.0 (Collision detection)
  - PongGameV4.0 (Lives lost, lives per game, score)
  - PongGameV5.0 (Tournament functionality)
  - PongGameV6.0 (Player class – array, no statistics)
  - PongGameV7.0 (Player class – array, with statistics)
  - PongGameV8.0 (JOptionPane for I/O)

Idea is based on Reas and Fry (2014) example

PongGame V8.0

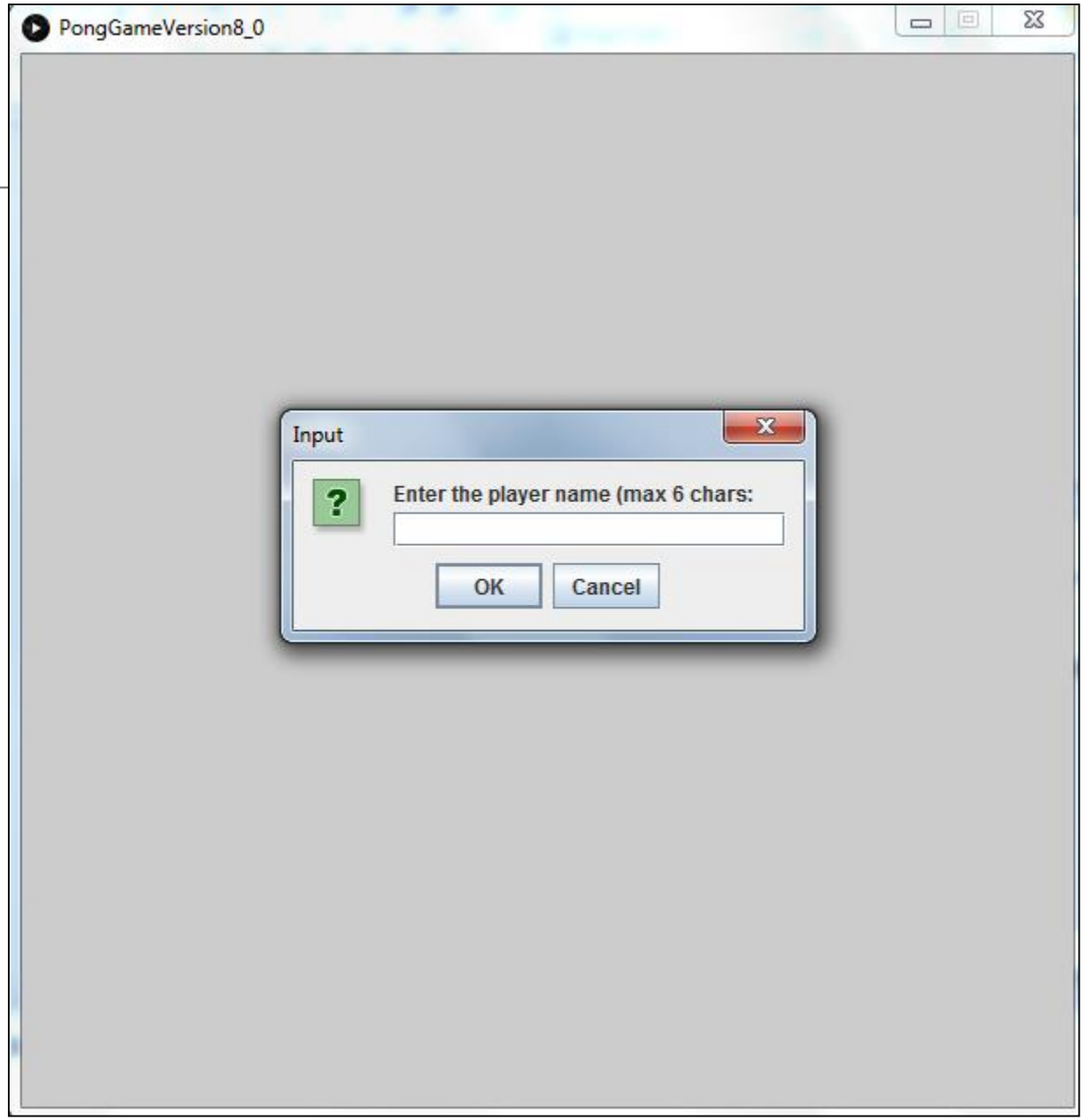
Player decides how many games of Pong they would like to play in their tournament.



PongGame V8.0

Enter player name.

> 6 chars, pong truncates the String.

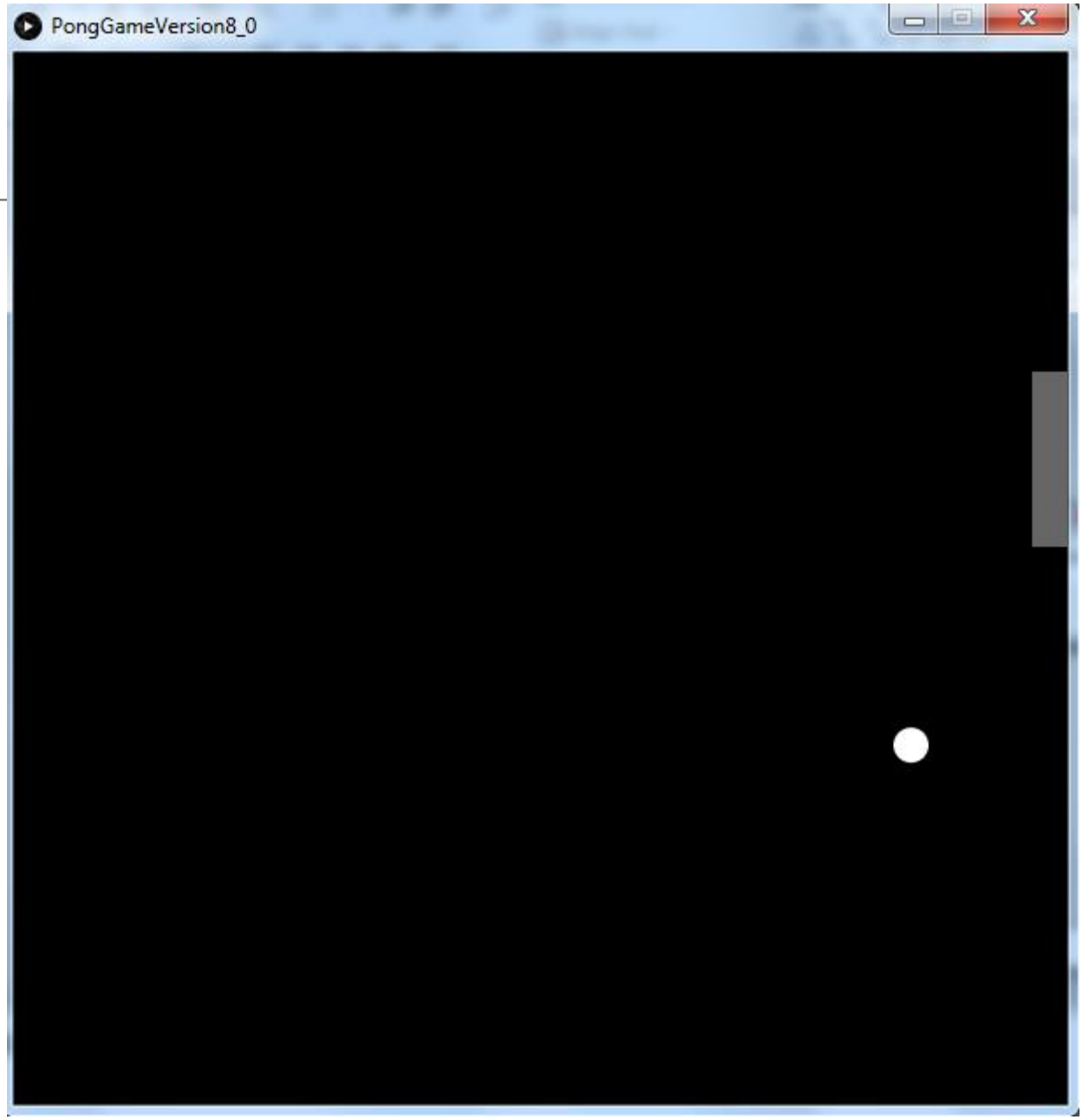


PongGame V8.0

When the ball is hit by the paddle → score increased by 1.

When the paddle misses the ball → a life is lost.

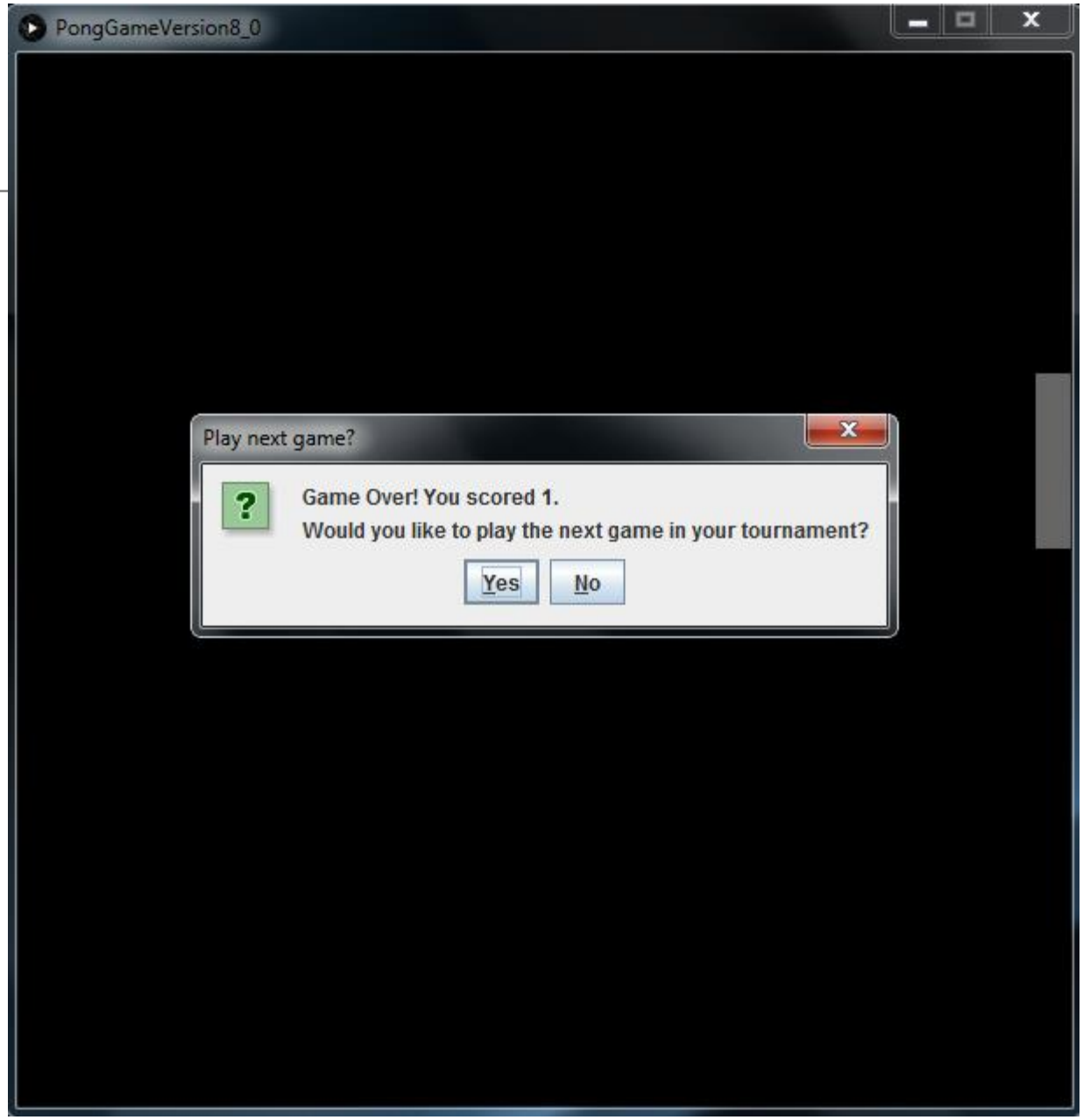
3 lives in a game.



PongGame V8.0

When a game ends and there are more games left to play in the tournament:

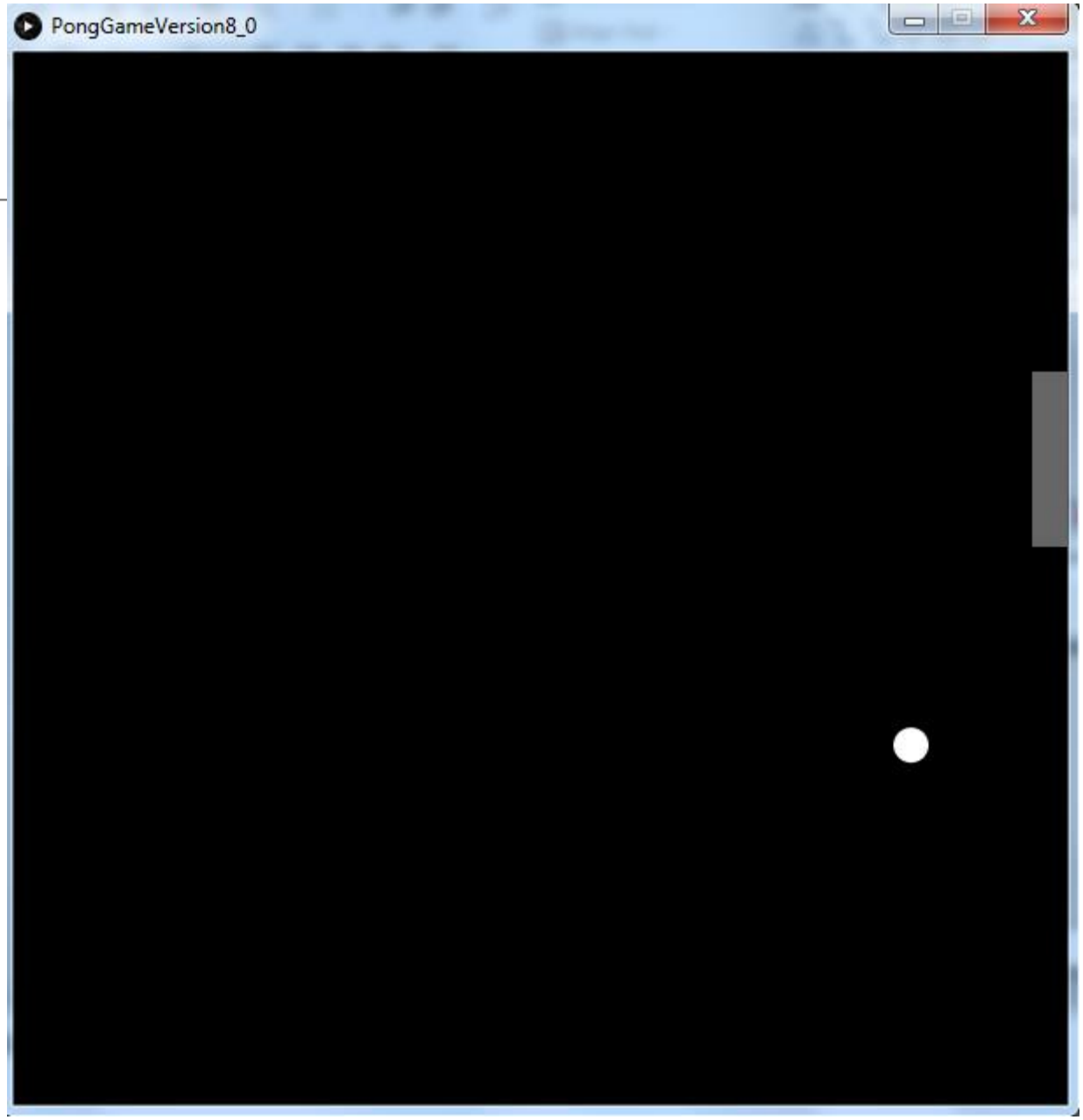
- Score is displayed.
- Player is asked if they want to continue with the tournament.



PongGame V8.0

If the player continues with the tournament:

- Game score is stored in an array.
- A new game is started (i.e. number of lives and score is reset to zero).

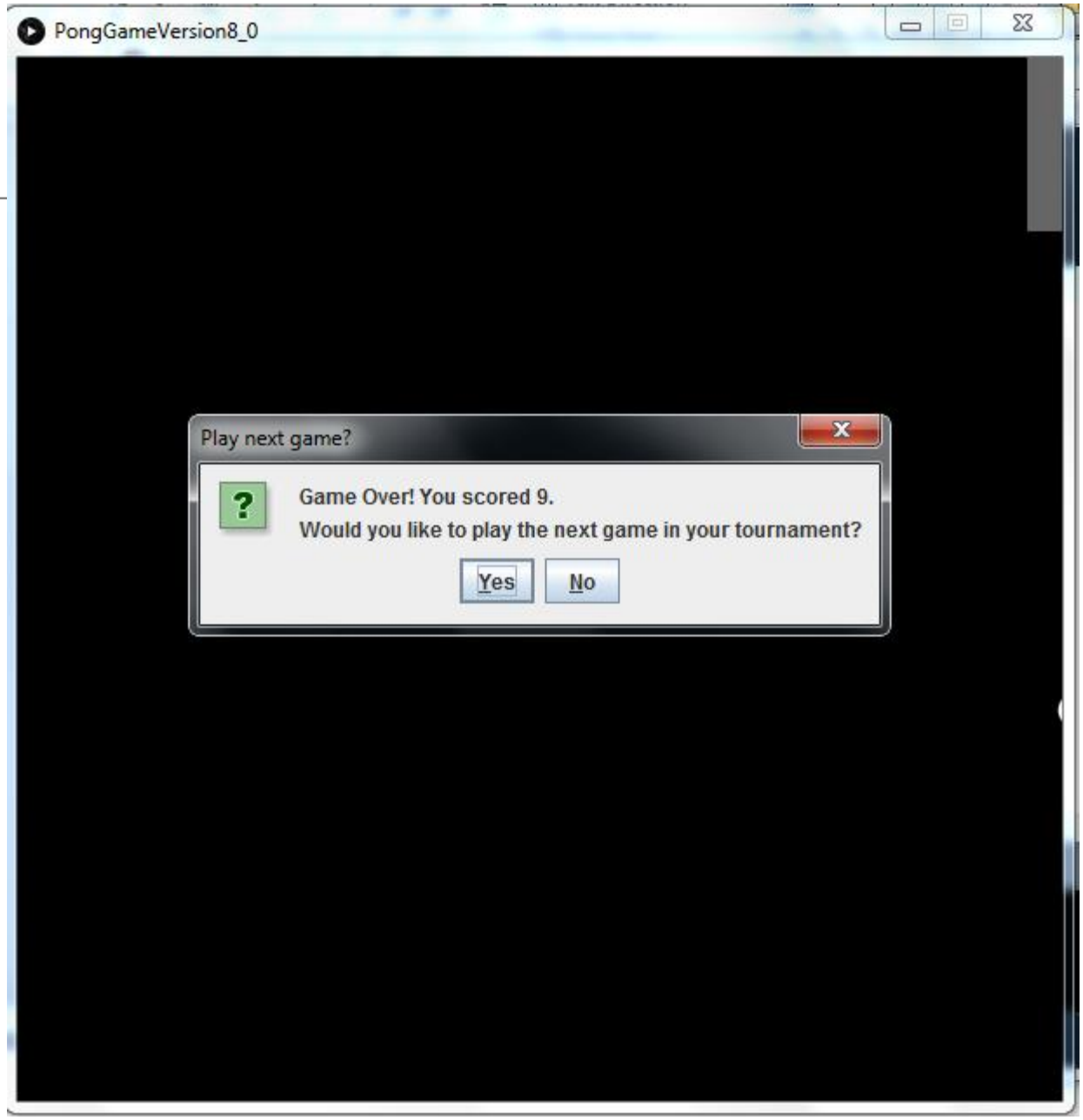


PongGame V8.0

Player is shown their score and asked if they wish to continue.

If they choose yes:

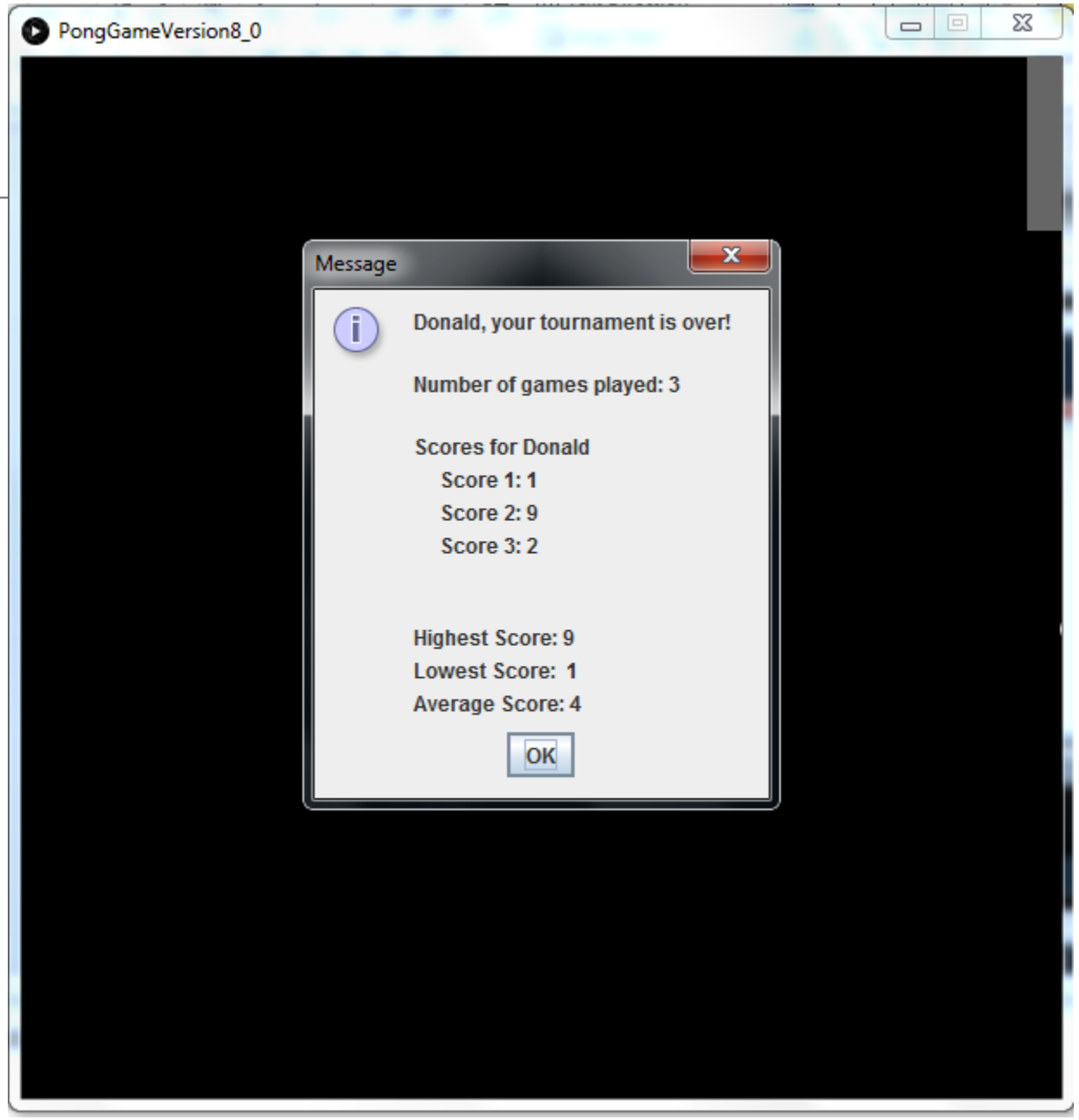
- Their score is stored in an array.
- A new game is started.



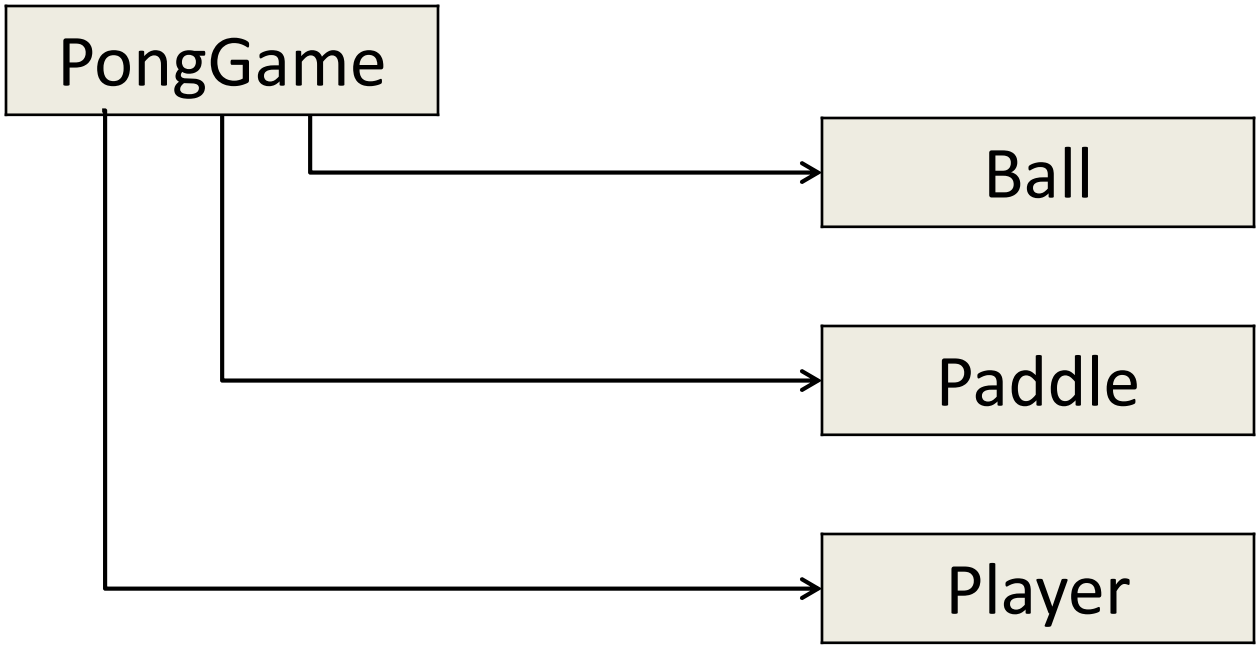
PongGame V8.0

When a game ends and NO more games are left in the tournament:

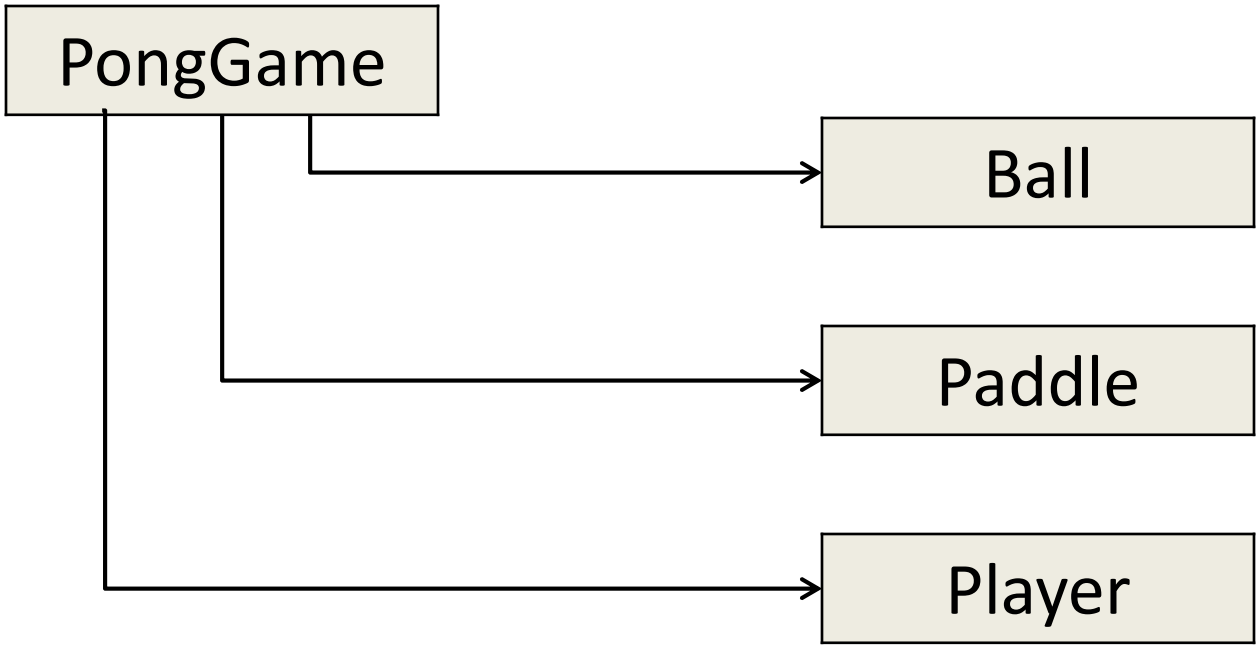
- Player name and score is displayed for each game in the tournament.
- Tournament statistics are also displayed (i.e. highest, lowest and average score).



PongGameV8.0

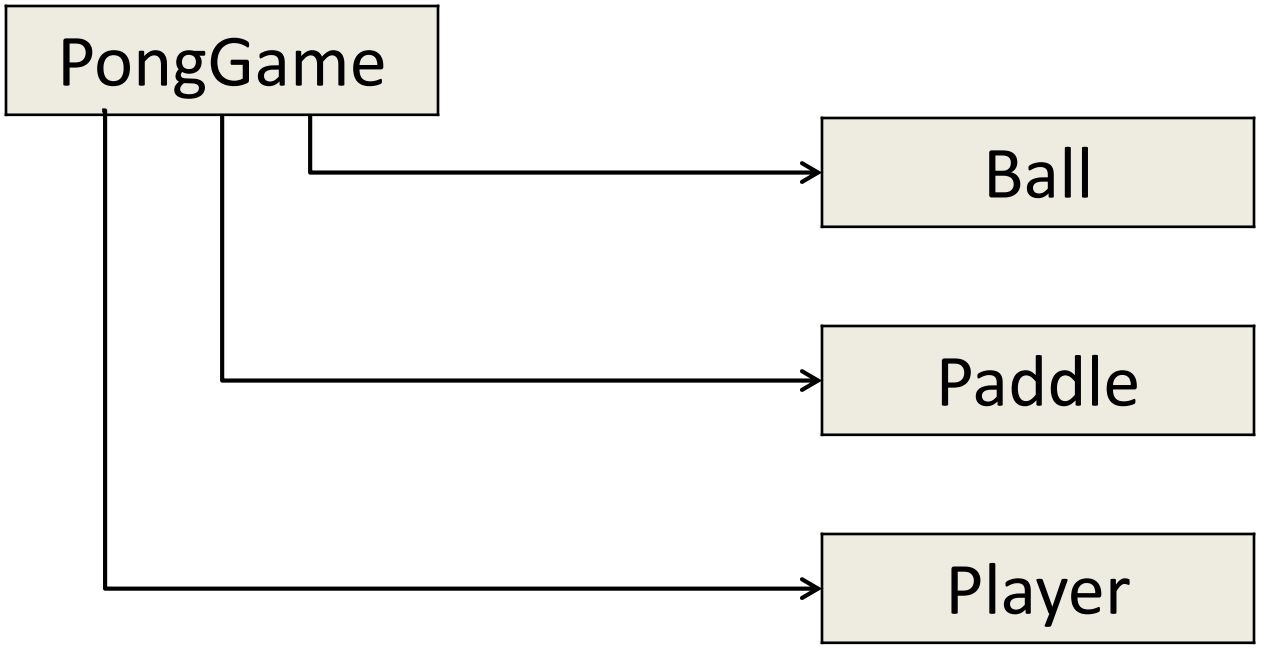


PongGameV8.0



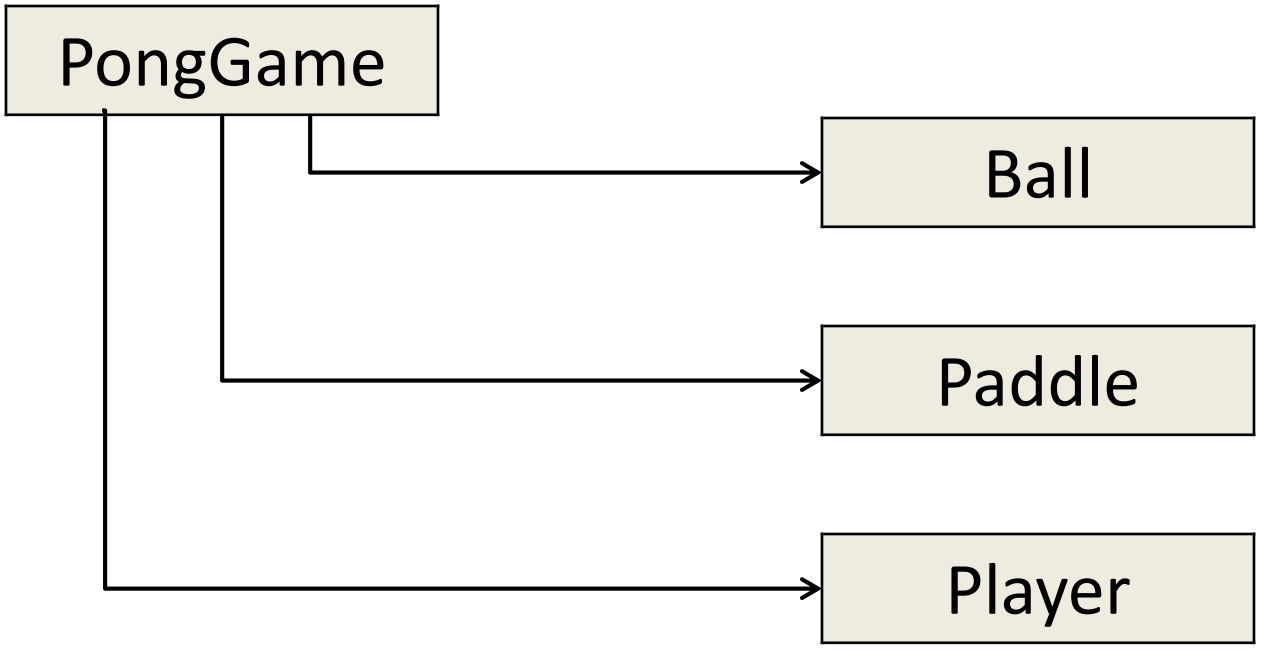
PongGame - has the setup() and draw() methods. It starts the game, handles player input, manages collision detection between the Ball and the Paddle, ends the game and outputs the player statistics.

PongGameV8.0



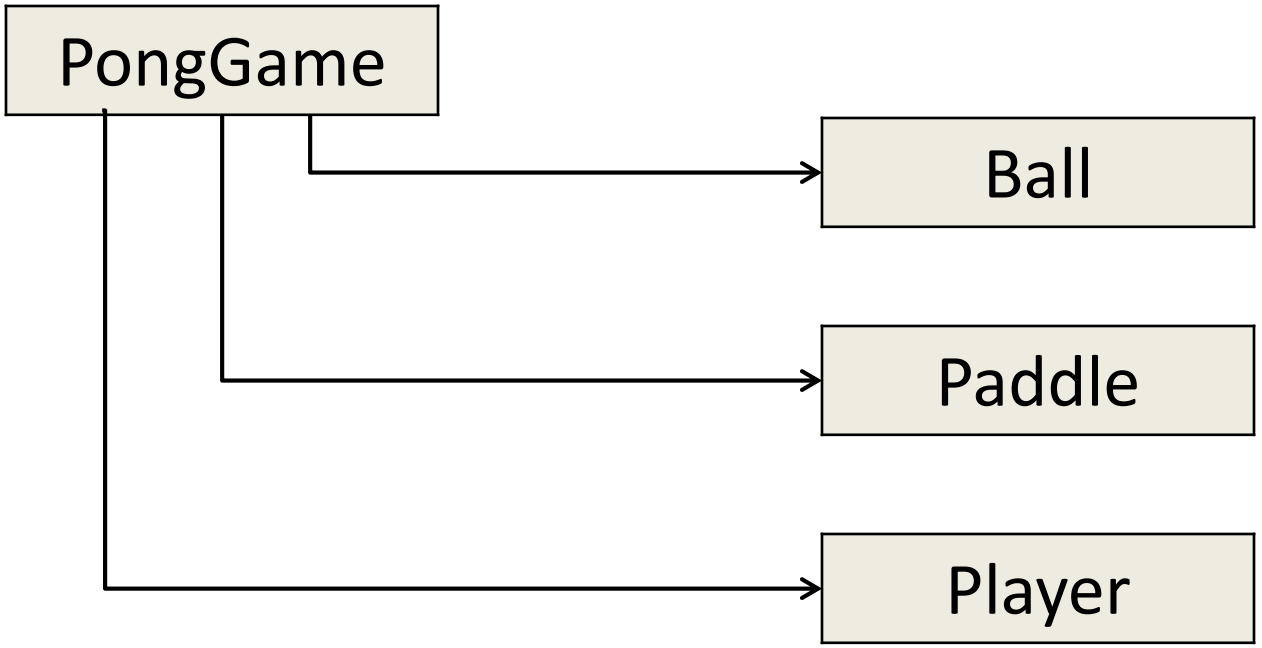
Ball - controls the location, size and speed of the ball. This class updates the ball and displays it at the updated location.

PongGameV8.0



Paddle - controls the location and size of the paddle. This class updates the paddle location and displays it at the updated location.

PongGameV8.0



Player - stores the player name and the score for each game in the tournament. It calculates the statistics for the games in the tournament

Topics list

- Overview of PongGameV8.0
- Developing:
  - PongGameV1.0 (Ball class)
  - PongGameV2.0 (Paddle class)
  - PongGameV3.0 (Collision detection)
  - PongGameV4.0 (Lives lost, lives per game, score)
  - PongGameV5.0 (Tournament functionality)
  - PongGameV6.0 (Player class – array, no statistics)
  - PongGameV7.0 (Player class – array, with statistics)
  - PongGameV8.0 (JOptionPane for I/O)

Idea is based on Reas and Fry (2014) example

Demo of
Pong Game V1.0

Classes in the PongGameV1.0

PongGame
ball
setup() ***draw()***

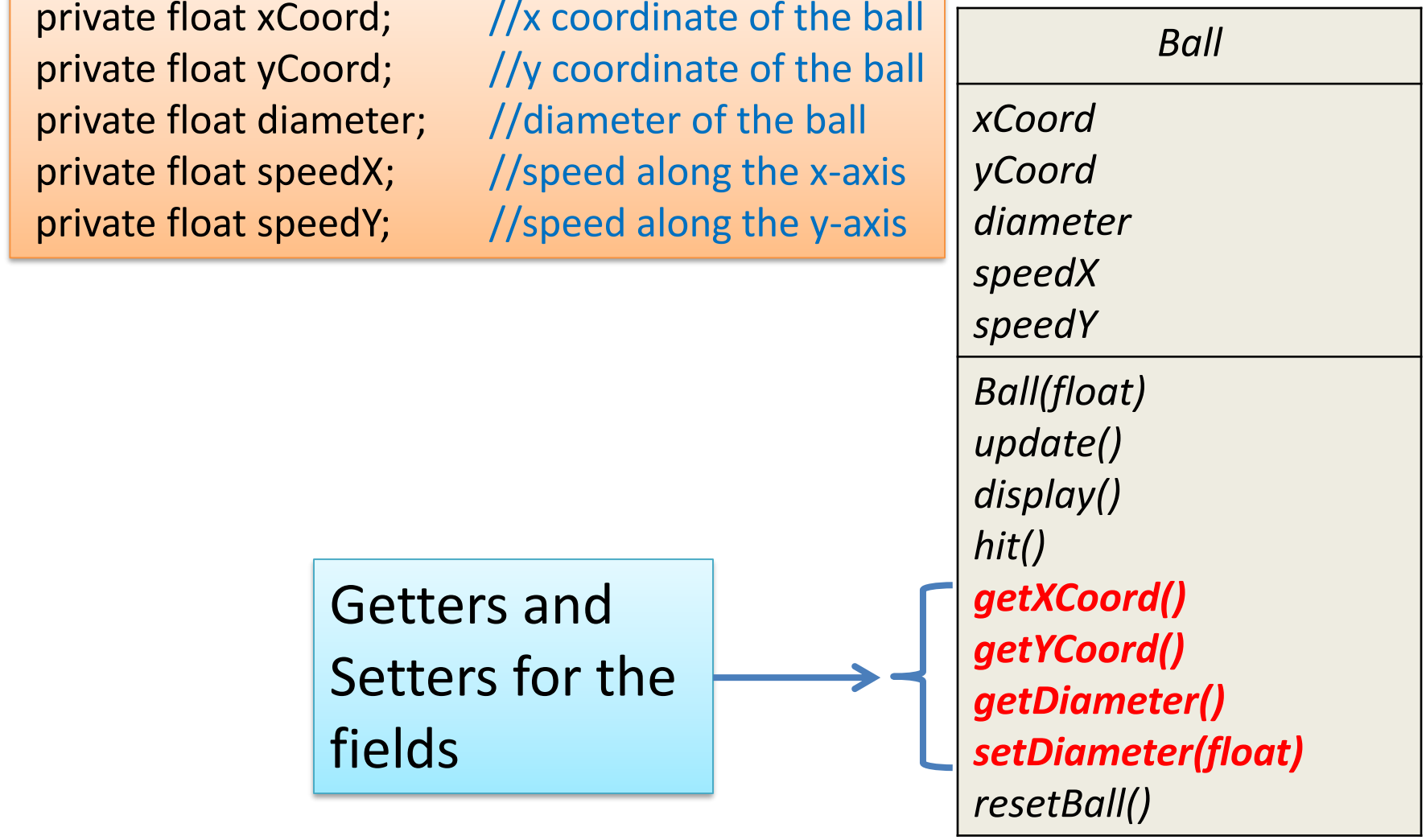
Ball
xCoord *yCoord* *diameter* *speedX* *speedY*
Ball(float) ***update()*** ***display()*** *hit()* *getXCoord()* *getYCoord()* *getDiameter()* *setDiameter(float)* *resetBall()*

The setup() method calls the Ball(float) constructor.

The draw() method calls the update() and display() methods in the Ball class.

Ball Class – instance fields

```
private float xCoord;        //x coordinate of the ball
private float yCoord;        //y coordinate of the ball
private float diameter;      //diameter of the ball
private float speedX;        //speed along the x-axis
private float speedY;        //speed along the y-axis
```



Ball Class – getters

```
public float getXCoord(){
   return xCoord;
}

public float getYCoord(){
   return yCoord;
}

public float getDiameter(){
   return diameter;
}
```

Ball
xCoord *yCoord* *diameter* *speedX* *speedY*
Ball(float) *update()* *display()* *hit()* ***getXCoord()*** ***getYCoord()*** ***getDiameter()*** ***setDiameter(float)*** *resetBall()*

Ball Class – setter

```
public void setDiameter(float diameter){
   //The ball diameter must be between 20 and height/6 (inclusive)
    if ((diameter >= 20) && (diameter <= height/6)){
      this.diameter = diameter;
    }
    else {
      // If an invalid diameter is passed as a parameter, a default of 20 is imposed.
      // With this animation, if we do not supply a default value for the diameter,
      // a ball may not be drawn on the display window.  Important note: it is not
      // always appropriate to provide a default value at sette) level; this will
      //depend on your design.
      this.diameter = 20;
    }
  }
```

display() method

```
public void display(){
  fill(255);
  noStroke();
  ellipse(xCoord, yCoord, diameter, diameter);
}
```

Draws a white ball, with no outline on the display window.

Ball
xCoord *yCoord* *diameter* *speedX* *speedY*
Ball(float) *update()* ***display()*** *hit()* *getXCoord()* *getYCoord()* *getDiameter()* *setDiameter(float)* *resetBall()*

private helper method

```
private void resetBall(){
   xCoord = 0;
   yCoord = random(height);
   speedX = random(3, 5);
   speedY = random(-2, 2);
}
```

A private helper method is a method that is private to the class you are in i.e. you are not allowed to use it outside of the current class.

The **resetBall** method is used by the **Ball** constructor and the **update** method.

Ball
xCoord *yCoord* *diameter* *speedX* *speedY*
Ball(float) *update()* *display()* *hit()* *getXCoord()* *getYCoord()* *getDiameter()* *setDiameter(float)* ***resetBall()***

A note on random

```
private void resetBall(){
   xCoord = 0;
   yCoord = random(height);
   speedX = random(3, 5);
   speedY = random(-2, 2);
}
```

random (high)

returns a random float between zero (inclusive) and high (exclusive).

random (low, high)

returns a random float between low (inclusive) and high (exclusive).

Ball constructor

```
public Ball(float diameter){
    setDiameter(diameter);
    resetBall();
}
```

This constructor takes in the diameter of the ball and uses the **setDiameter** setter method to update the diameter instance field.

The private helper method, **resetBall** is called to set up the xCoord with zero and yCoord, speedX and speedY with random values.

Ball
xCoord *yCoord* *diameter* *speedX* *speedY*
Ball(float) *update()* *display()* *hit()* *getXCoord()* *getYCoord()* *getDiameter()* *setDiameter(float)* *resetBall()*

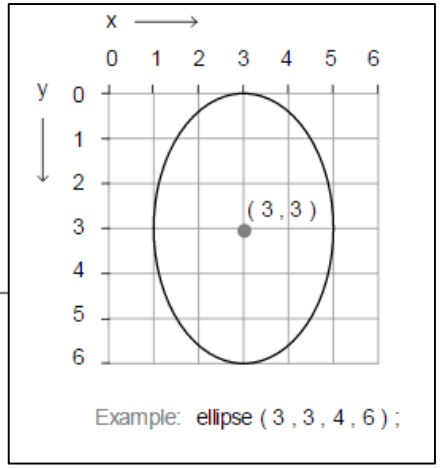
update() method

```
public boolean update(){
   boolean lifeLost = false;

   //update ball coordinates
   xCoord = xCoord + speedX;
   yCoord = yCoord + speedY;

  //reset position if ball leaves the screen
  if (xCoord > width + diameter/2){
      resetBall();
      lifeLost = true;
  }
```

```
  // If ball hits the left edge of the display
  // window, change direction of xCoord
  if (xCoord < diameter/2)
     xCoord = diameter/2;
     speedX = speedX * -1;
  }

  // If ball hits top or bottom of the display
  //  window, change direction of yCoord
  if (yCoord > height - diameter/2){
     yCoord = height - diameter/2;
     speedY = speedY * -1;
  }
  else if (yCoord < diameter/2){
     yCoord = diameter/2;
     speedY = speedY * -1;
  }
  return lifeLost;
}
```

The update method changes the ball position. It returns true if the ball goes off the screen (i.e. a life was lost)

update() method - explained




```
//reset position if ball leaves the screen
if (xCoord > width + diameter/2){
   resetBall();
   lifeLost = true;
}
```

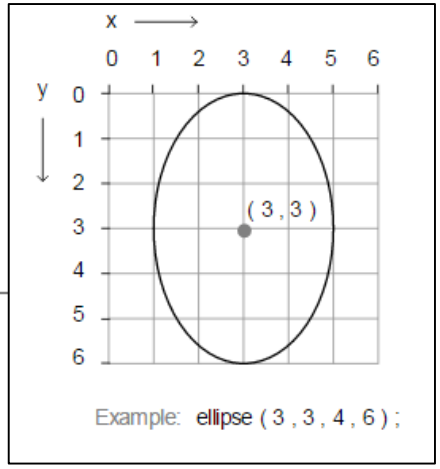
(width + diameter/2)

In this check, we add diameter/2 onto the width of the window so that the ball is completely off the screen.

```
// If ball hits the left edge of the display
// window, change direction of xCoord
if (xCoord < diameter/2)
   xCoord = diameter/2;
   speedX = speedX * -1;
}
```

If the xCoord is less than the radius of the circle, the circle has hit the left side → reset the xCoord to the radius of the circle and reverse the speedX variable by multiplying by -1.

update() method - explained




```
    // If ball hits top or bottom of the display
    //  window, change direction of yCoord
    if (yCoord > height - diameter/2){
       yCoord = height - diameter/2;
       speedY = speedY * -1;
    }
    else if (yCoord < diameter/2){
       yCoord = diameter/2;
       speedY = speedY * -1;
    }

    // Return whether a life was lost in the
    // game → true means a life was lost.
    return lifeLost;
}
```

The yCoord is investigated in this section to see if the top or bottom of the screen was hit.

The ball has hit the top if:

(yCoord < diameter/2)

The ball has hit the bottom if:

(yCoord > height - diameter/2)

hit() method

```
public void hit(){
    speedX = speedX * -1;
    xCoord = xCoord + speedX;
}
```

We are not using this method in this version of Pong, but we are preparing our class for collision detection in V3.0.

This method changes the ball direction when it hits the paddle and bump it back to the edge of the paddle.

Ball
xCoord *yCoord* *diameter* *speedX* *speedY*
Ball(float) *update()* *display()* ***hit()*** *getXCoord()* *getYCoord()* *getDiameter()* *setDiameter(float)* *resetBall()*

PongGame V1.0

```
Ball ball;

void setup(){
  size(600,600);
  noCursor();
  //setting up the ball with hard-coded sizes.
  ball = new Ball(20.0);
}

void draw(){
  background(0);
  //Update the ball position and display it.
  ball.update();
  ball.display();
}
```

PongGame
ball
setup()   ***draw()***

Topics list

- Overview of PongGameV8.0
- Developing:
  - PongGameV1.0 (Ball class)
  - PongGameV2.0 (Paddle class)
  - PongGameV3.0 (Collision detection)
  - PongGameV4.0 (Lives lost, lives per game, score)
  - PongGameV5.0 (Tournament functionality)
  - PongGameV6.0 (Player class – array, no statistics)
  - PongGameV7.0 (Player class – array, with statistics)
  - PongGameV8.0 (JOptionPane for I/O)

Idea is based on Reas and Fry (2014) example

Demo of
Pong Game V2.0

Classes in the PongGameV2.0

PongGame
ball *paddle*
setup() ***draw()***

The setup() method calls the Ball(float) and Paddle (int int) constructor.

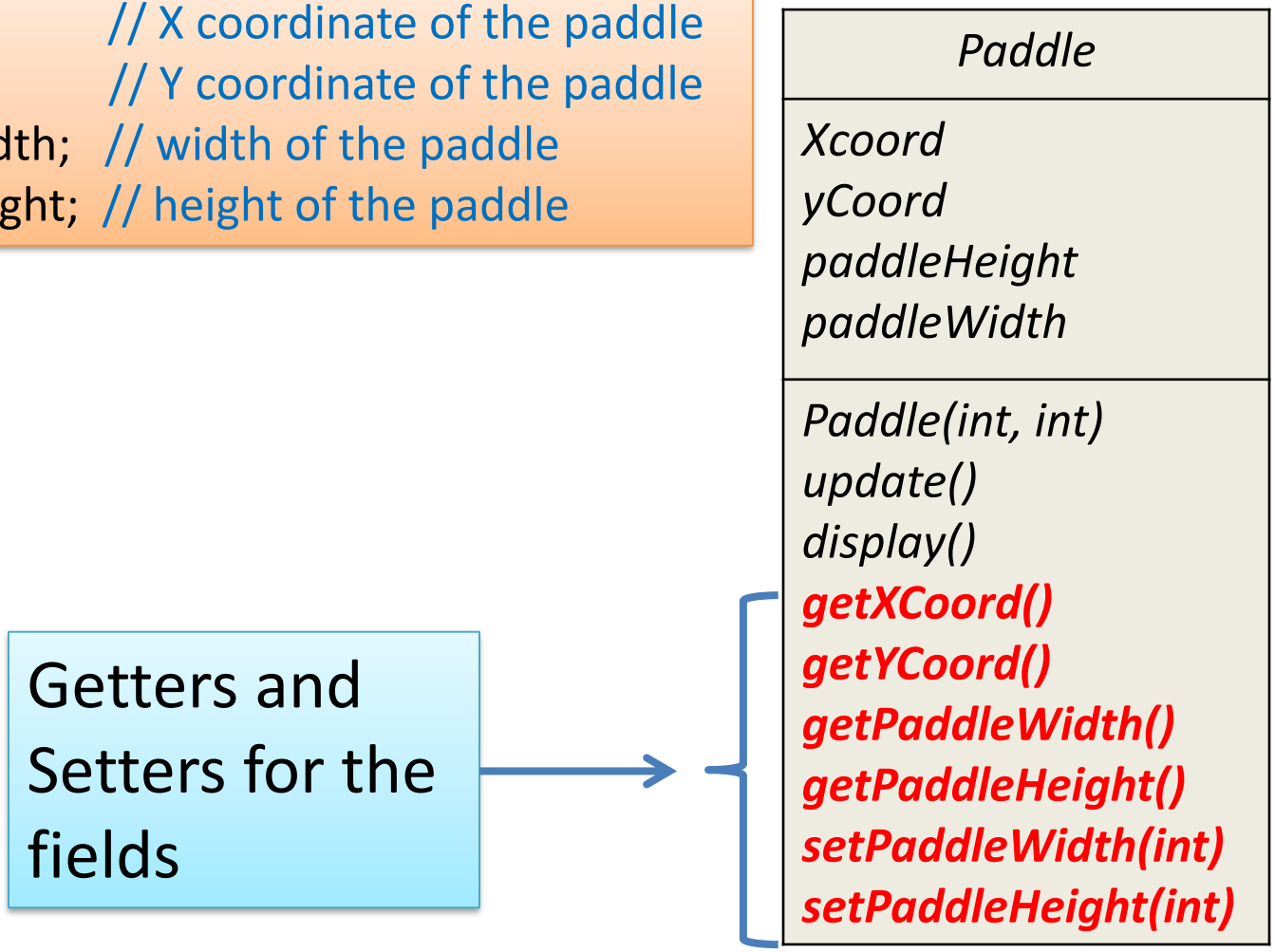
The draw() method calls the update() and display() methods in both the Ball and Paddle class.

Paddle
Xcoord *yCoord* *paddleHeight* *paddleWidth*
Paddle(int, int) ***update()*** ***display()*** *getXCoord()* *getYCoord()* *getPaddleWidth()* *getPaddleHeight()* *setPaddleWidth(int)* *setPaddleHeight(int)*

Ball
xCoord *yCoord* *diameter* *speedX* *speedY*
Ball(float) ***update()*** ***display()*** *hit()* *getXCoord()* *getYCoord()* *getDiameter()* *setDiameter(float)* *resetBall()*

Paddle Class – instance fields

```
private int xCoord;          // X coordinate of the paddle
private int yCoord;          // Y coordinate of the paddle
private int paddleWidth;     // width of the paddle
private int paddleHeight;    // height of the paddle
```



Paddle Class – getters

```
public int getXCoord(){
   return xCoord;
}

public int getYCoord(){
   return yCoord;
}

public int getPaddleWidth(){
   return paddleWidth;
}

public int getPaddleHeight(){
   return paddleHeight;
}
```

Paddle
Xcoord *yCoord* *paddleHeight* *paddleWidth*
Paddle(int, int) *update()* *display()* ***getXCoord()*** ***getYCoord()*** ***getPaddleWidth()*** ***getPaddleHeight()*** *setPaddleWidth(int)* *setPaddleHeight(int)*

Paddle Class – setPaddleWidth(int)

```
public void setPaddleWidth(int paddleWidth){
    //The paddle width must be between 10 and
    //width/2 (inclusive)
    if ((paddleWidth >= 20) && (paddleWidth <= width/2)){
        this.paddleWidth = paddleWidth;
    }
    else{
    // If an invalid width is passed as a parameter, a default
    // width of 20 is imposed.  With this animation, if we do
    // not supply a default value for the width, a paddle
    //may not be drawn on the display window.  Important
    // note: it is not always appropriate to provide a default
    // value at setter level; this will depend on your
    //design.
        this.paddleWidth = 20;
    }
}
```

Paddle
Xcoord *yCoord* *paddleHeight* *paddleWidth*
Paddle(int, int) *update()* *display()* *getXCoord()* *getYCoord()* *getPaddleWidth()* *getPaddleHeight()* ***setPaddleWidth(int)*** *setPaddleHeight(int)*

Paddle Class – setPaddleHeight(int)

```
public void setPaddleHeight(int paddleHeight){
    //The paddle height must be between 50
    //and height/2 (inclusive)
    if ((paddleHeight >= 50) && (paddleHeight <= height/2)){
        this.paddleHeight = paddleHeight;
    }
    else{
    // If an invalid height is passed as a parameter, a default
    // height of 50 is imposed.  With this animation, if we do
    // not supply a default value for the height, a paddle
    // may not be drawn on the display window.  Important
    // note: it is not always appropriate to provide a default
    // value at setter level; this will depend on your design.
        this.paddleHeight = 50;
    }
  }
```

Paddle
Xcoord *yCoord* *paddleHeight* *paddleWidth*
Paddle(int, int) *update()* *display()* *getXCoord()* *getYCoord()* *getPaddleWidth()* *getPaddleHeight()* *setPaddleWidth(int)* ***setPaddleHeight(int)***

Paddle constructor

```
public Paddle(int paddleWidth, int paddleHeight)
{
    setPaddleWidth(paddleWidth);
    setPaddleHeight(paddleHeight);

    // the xCoordinate variable is set here and it stays
    // this value for duration of the program.
    xCoord = width - this.paddleWidth;

    // the yCoordinate variable is set here and changes
    // later in the program as the mouse moves on the
    // vertical plane.
    yCoord = height/2;
 }
```

Paddle
Xcoord *yCoord* *paddleHeight* *paddleWidth*
Paddle(int, int) *update()* *display()* *getXCoord()* *getYCoord()* *getPaddleWidth()* *getPaddleHeight()* *setPaddleWidth(int)* *setPaddleHeight(int)*

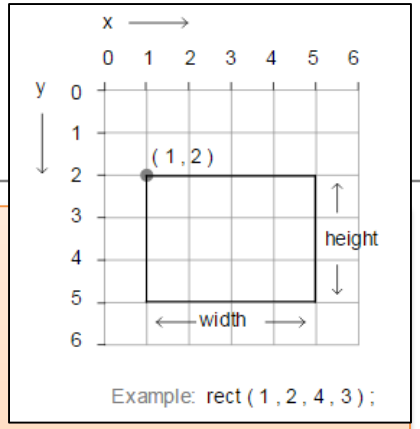
display() method

```
public void display(){
  fill(102);
  noStroke();
  rect(xCoord, yCoord, paddleWidth, paddleHeight);
}
```

Draws a gray paddle, with no outline on the display window.

Paddle
Xcoord *yCoord* *paddleHeight* *paddleWidth*
Paddle(int, int) *update()* ***display()*** *getXCoord()* *getYCoord()* *getPaddleWidth()* *getPaddleHeight()* *setPaddleWidth(int)* *setPaddleHeight(int)*

update() method




```
public void update()
{
   yCoord = mouseY - paddleHeight/2;

   //Reset yCoord if it's outside the window coordinates.
   if (yCoord < 0){
       yCoord = 0;
    }
    if (yCoord > (height - paddleHeight)){
       yCoord = height - paddleHeight;
    }
}
```

This method changes the vertical position of the paddle in line with the cursor.

Paddle
Xcoord *yCoord* *paddleHeight* *paddleWidth*
Paddle(int, int) ***update()*** *display()* *getXCoord()* *getYCoord()* *getPaddleWidth()* *getPaddleHeight()* *setPaddleWidth(int)* *setPaddleHeight(int)*

PongGame V2.0

```
Ball ball;
Paddle paddle;

void setup(){
  size(600,600);
  noCursor();
  //setting up ball and paddle with hard-coded sizes.
  ball = new Ball(20.0);
  paddle = new Paddle(20,100);
}

void draw(){
  background(0);
  //Update the paddle location in line with the cursor
  paddle.update();
  paddle.display();
  //Update the ball position and display it.
  ball.update();
  ball.display();
}
```

PongGame
Ball *paddle*
setup() ***draw()***

Questions?

References

- Reas, C. & Fry, B. (2014) Processing – A Programming Handbook for Visual Designers and Artists, 2nd Edition, MIT Press, London.

Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see http://creativecommons.org/licenses/by-nc/3.0/

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/