Game of Pong

Overview and starting development

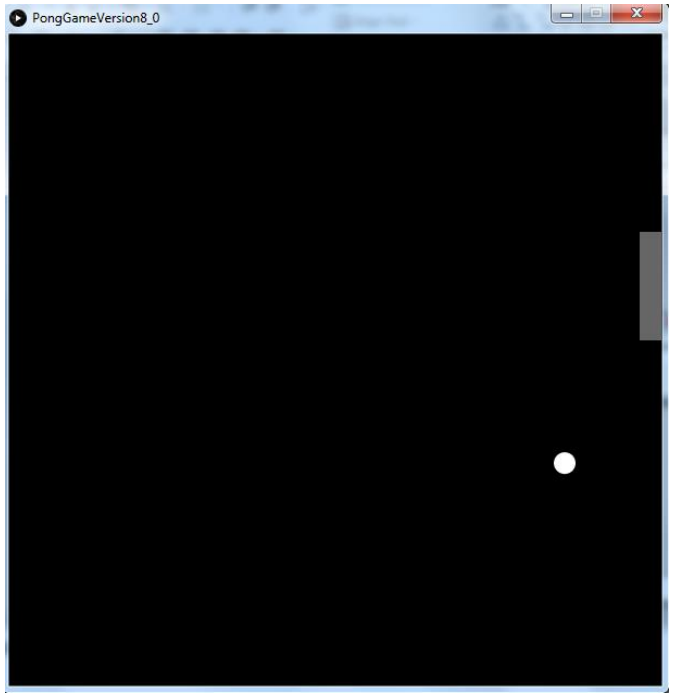
Produced by: Dr. Siobhán Drohan
Ms. Mairead Meagher

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

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

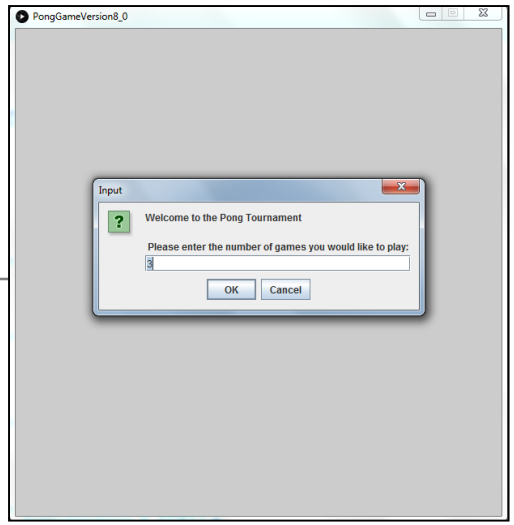
Topics list - **PONG**

- **Overview** of **PongGame**
- Developing PongGame
  - 9 versions (iterations) described with 4 sets of slides:
  - Set 1
    - V1.0 (Ball class)
    - V2.0 (Paddle class)
  - Set 2
    - V3.0 (Collision detection)
    - V4.0 (Lives lost, lives per game, score)
    - V5.0 (Tournament functionality)
  - Set 3
    - V6.0 (Player class – array, no statistics)
    - V7.0 (Player class – array, with statistics)
    - V8.0 (JOptionPane for I/O)
  - Set 4
    - V9.0 (Advanced Collision Detection)

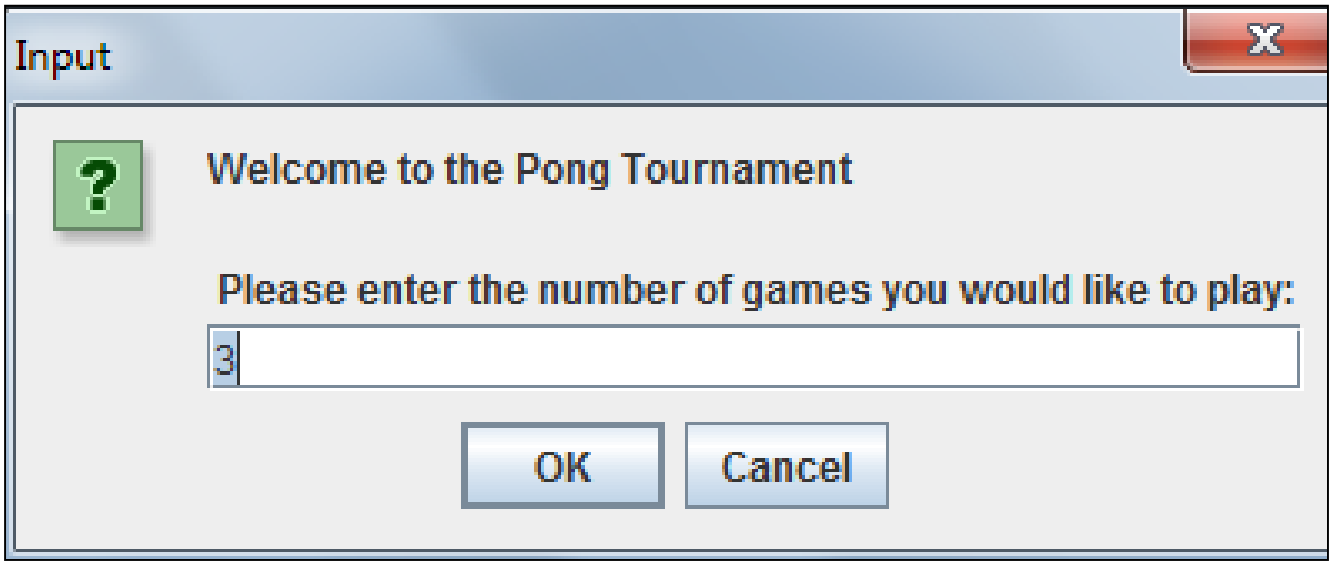


Idea is based on Reas and Fry (2014) example

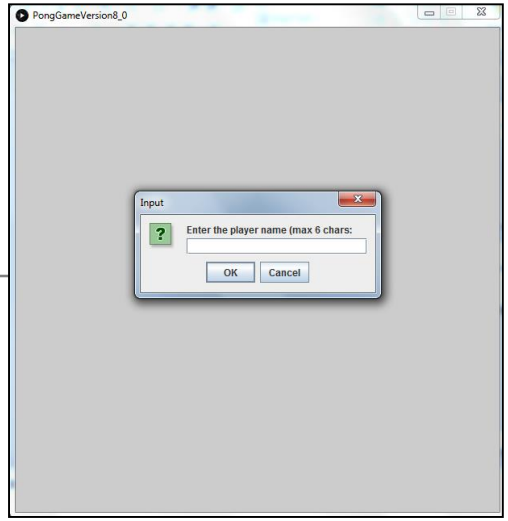
PongGame - Overview



Player decides **the NUMBER OF GAMES** of Pong they would like to play in their **tournament**.

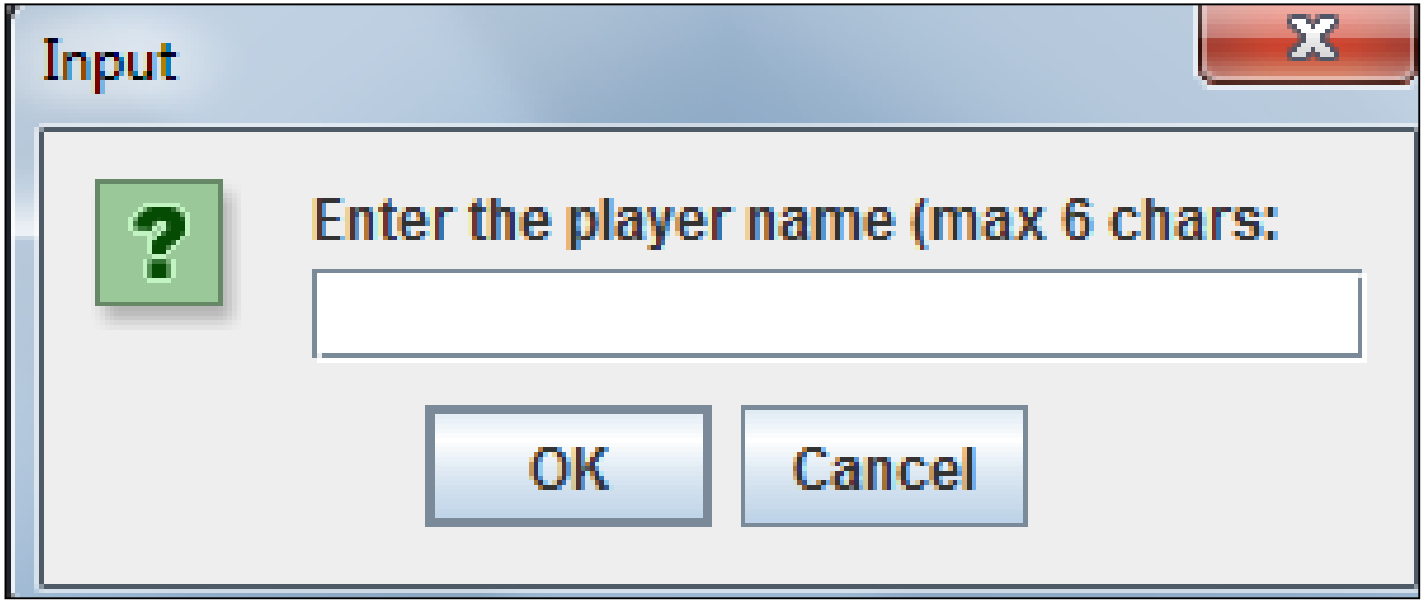


PongGame - Overview

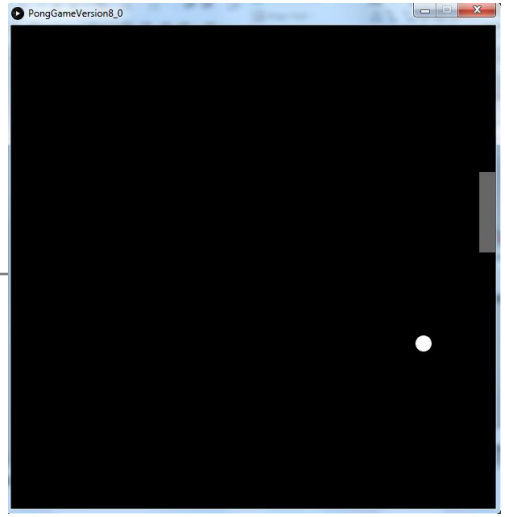


Enter PLAYER NAME

<= 6 chars, pong **truncates** the String



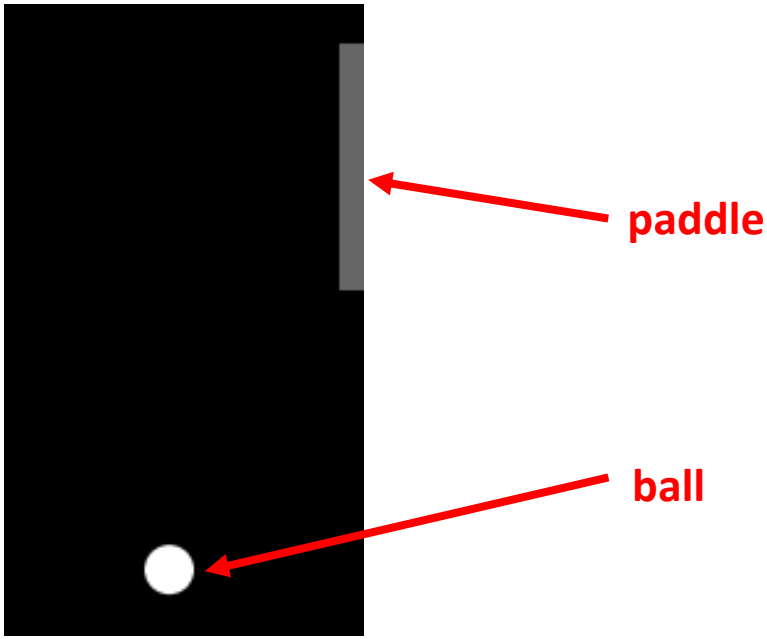
PongGame - Overview



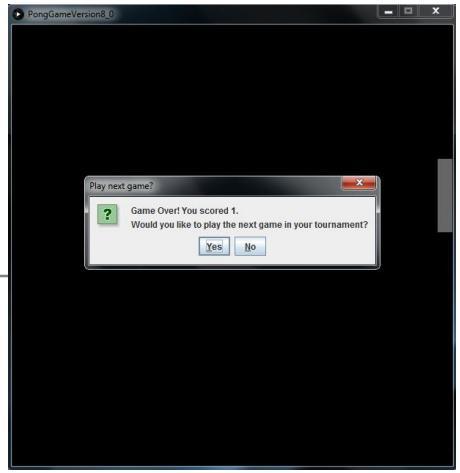
When the ball is **hit** by the paddle
→ **score increased by 1**.

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

Number of lives in a game
→ **3**

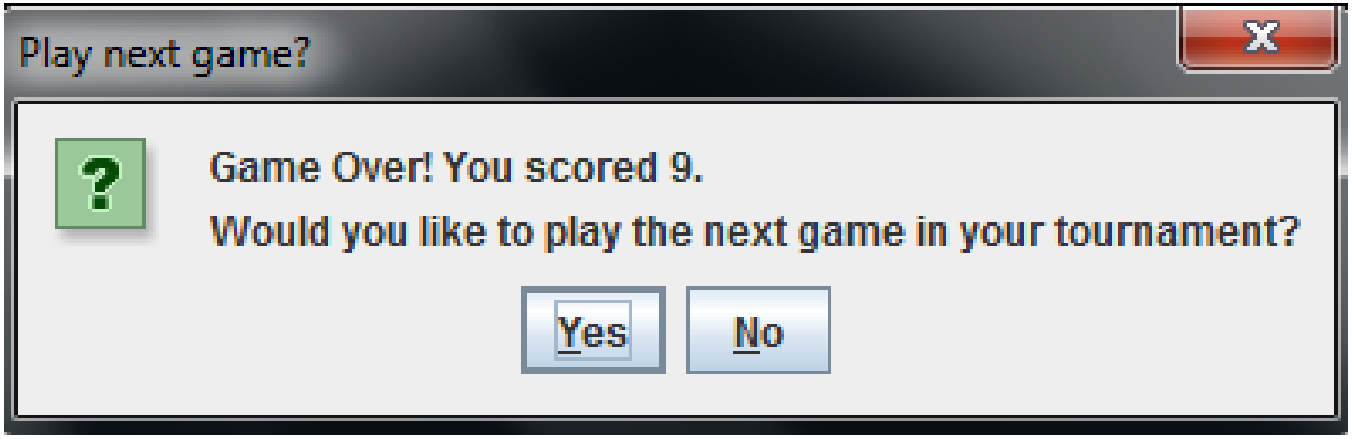


PongGame - Overview

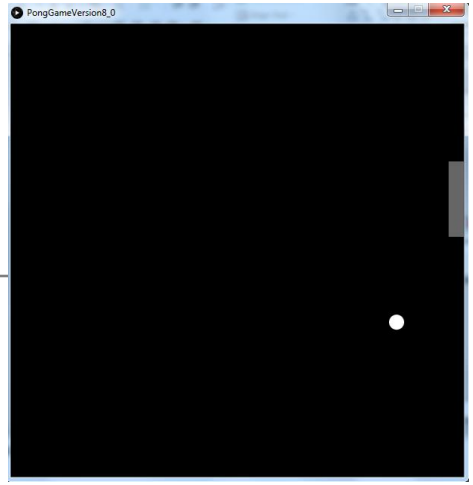


When a **game ends**
and there are more games left to play in the **tournament**:

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



PongGame - Overview

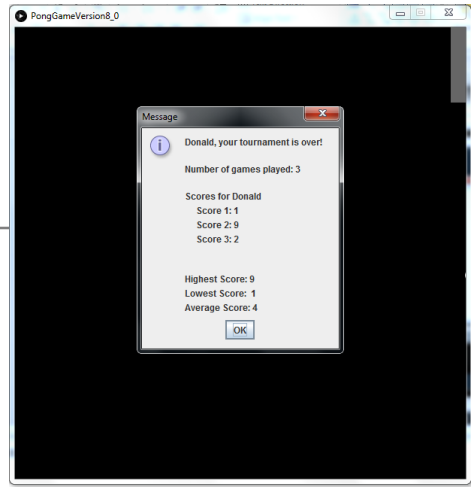


If the player **continues** with the tournament:

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

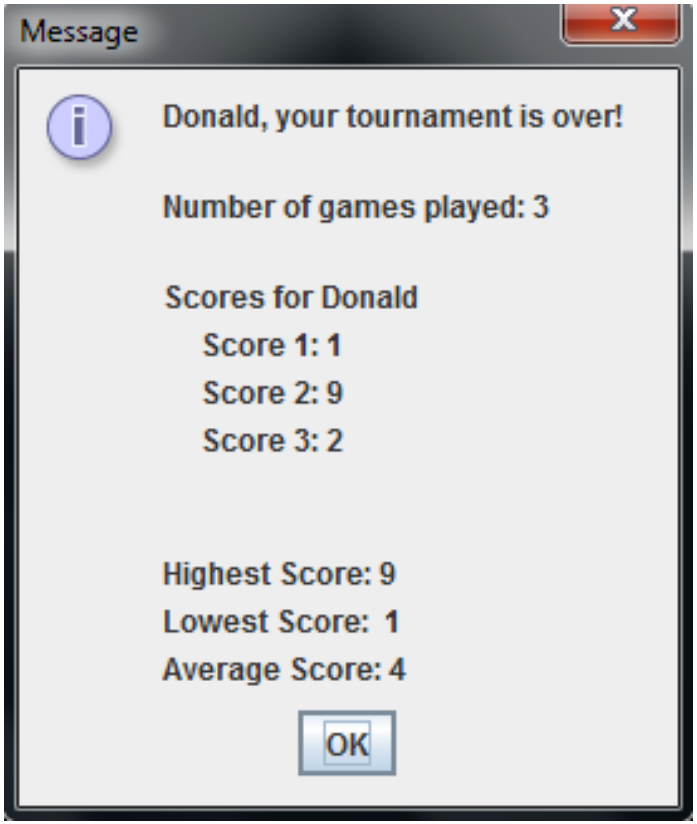
  - **number of lives lost → 0**
  - **Score → 0**

PongGame - Overview

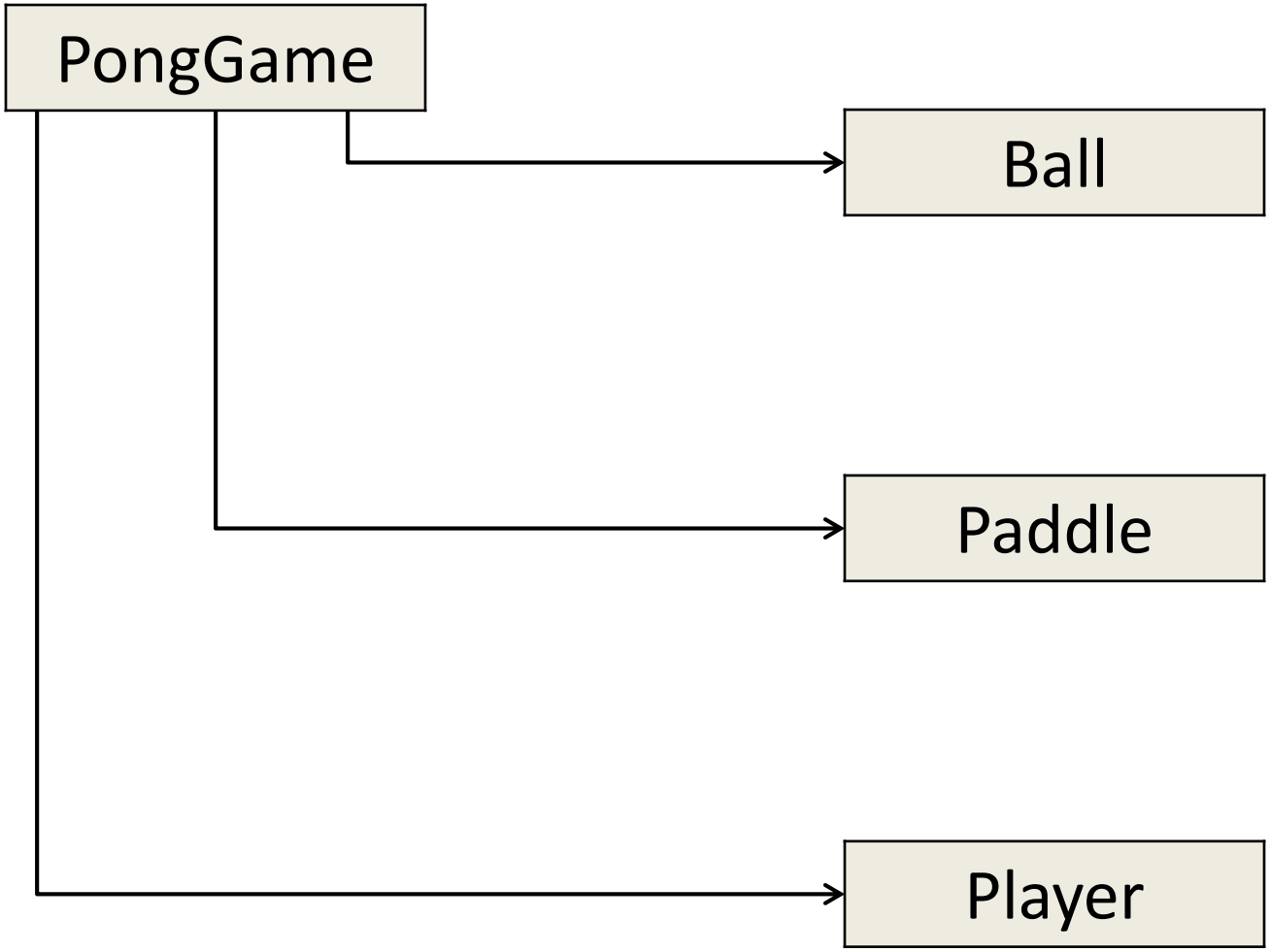


When a game ends
and **NO more games are left** in the tournament:

- **For each game in the tournament**
  Display player name and score
- **Display tournament statistics**
  (i.e. highest, lowest and average score).



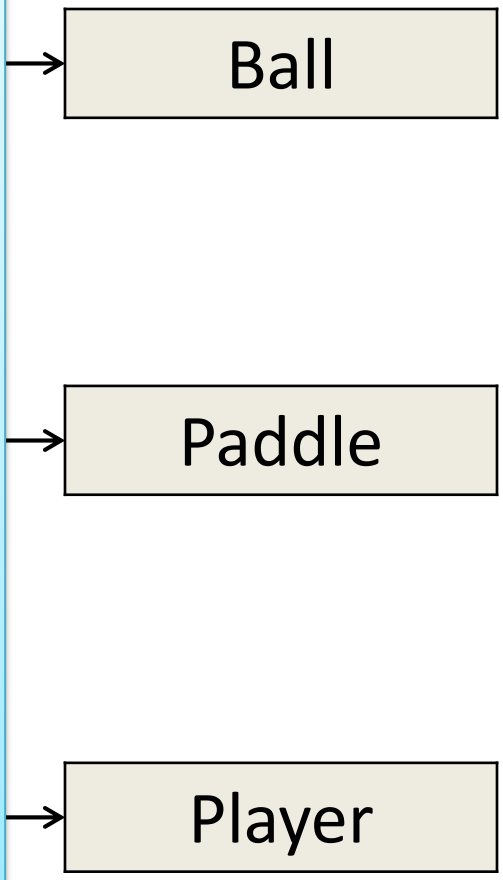
PongGame - Overview CLASSES



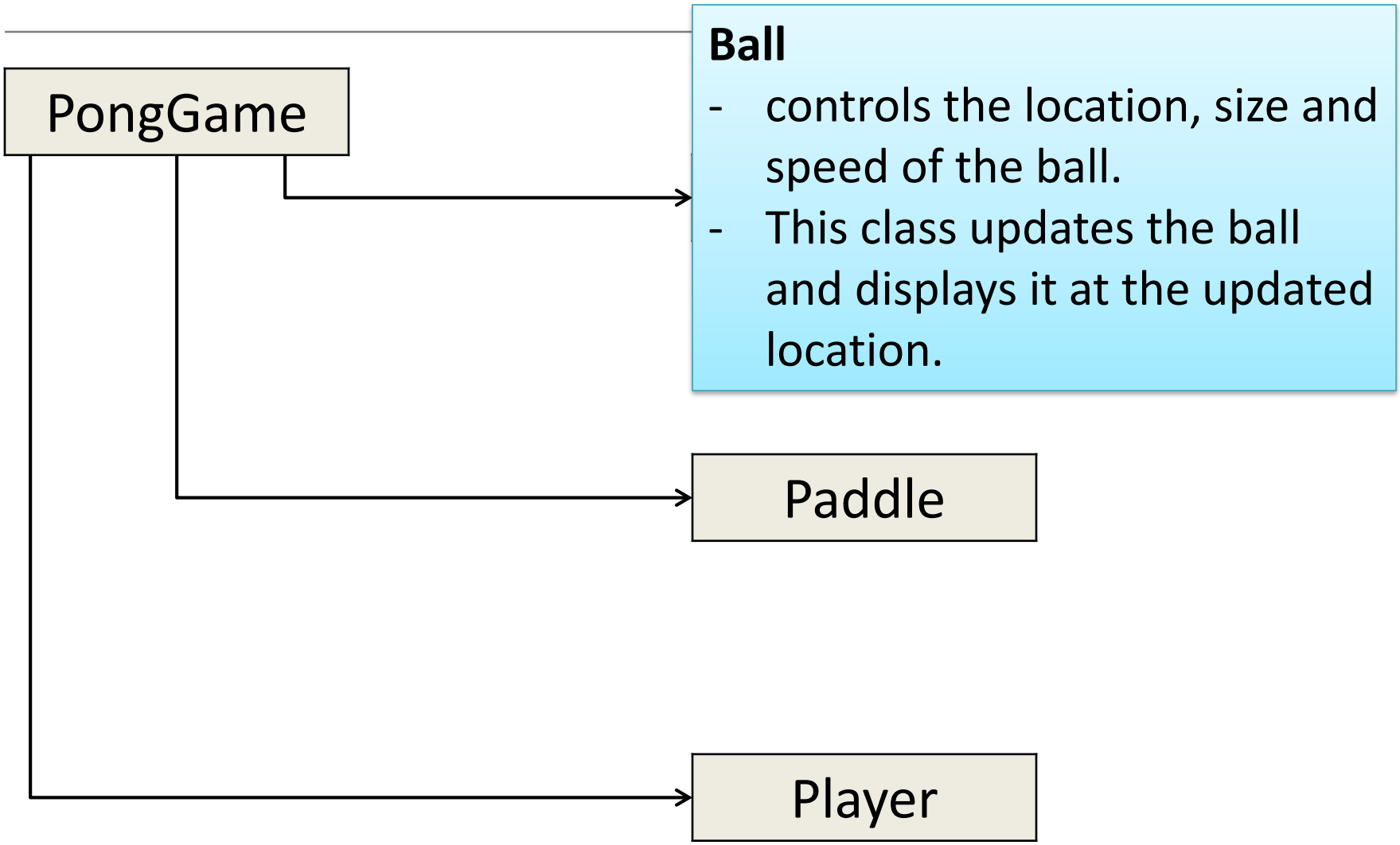
PongGame - Overview

PongGame

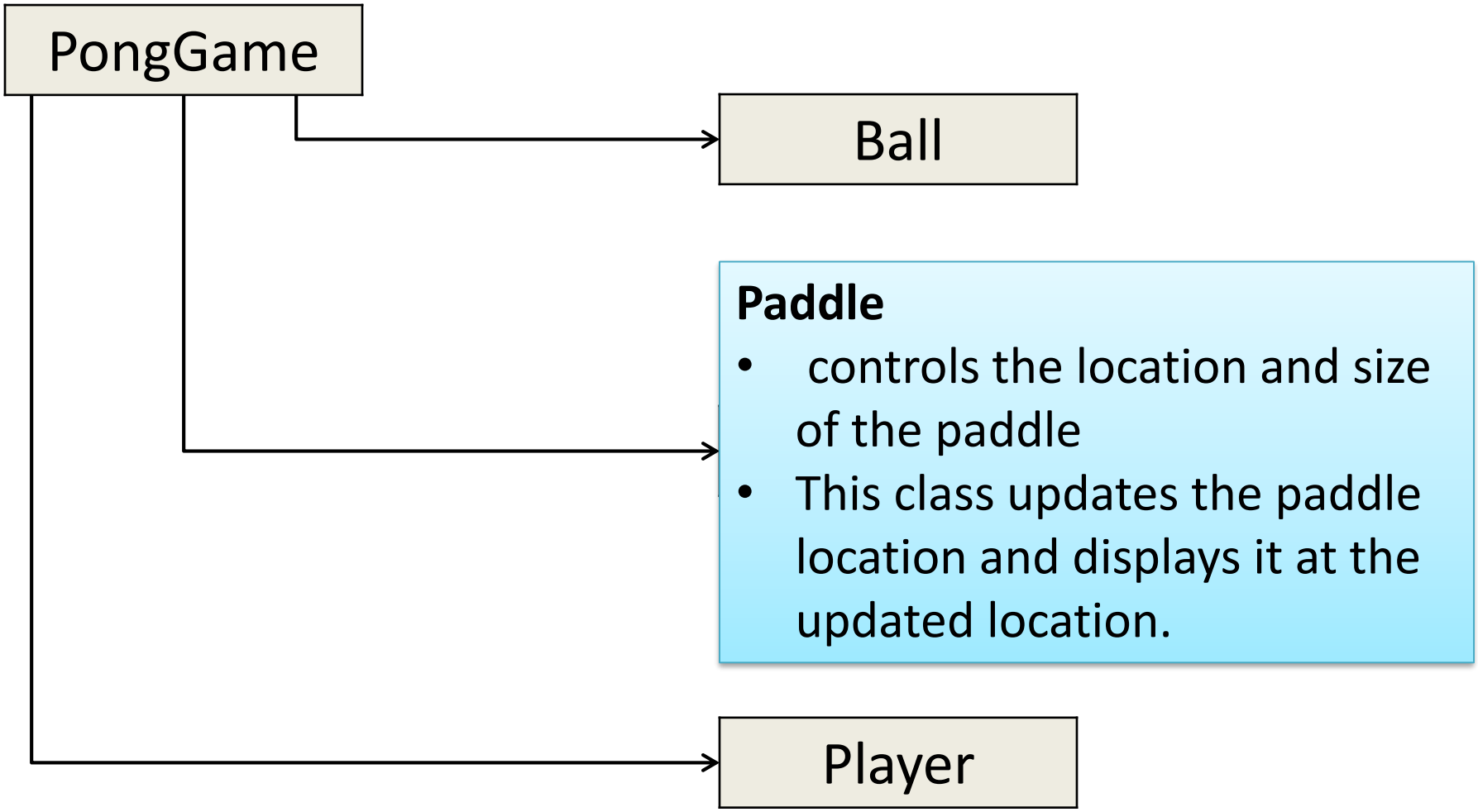
- has the **setup()** and **draw()** methods
- starts the game
- handles player input
- manages collision detection between the Ball and the Paddle,
- ends the game
- outputs the player statistics



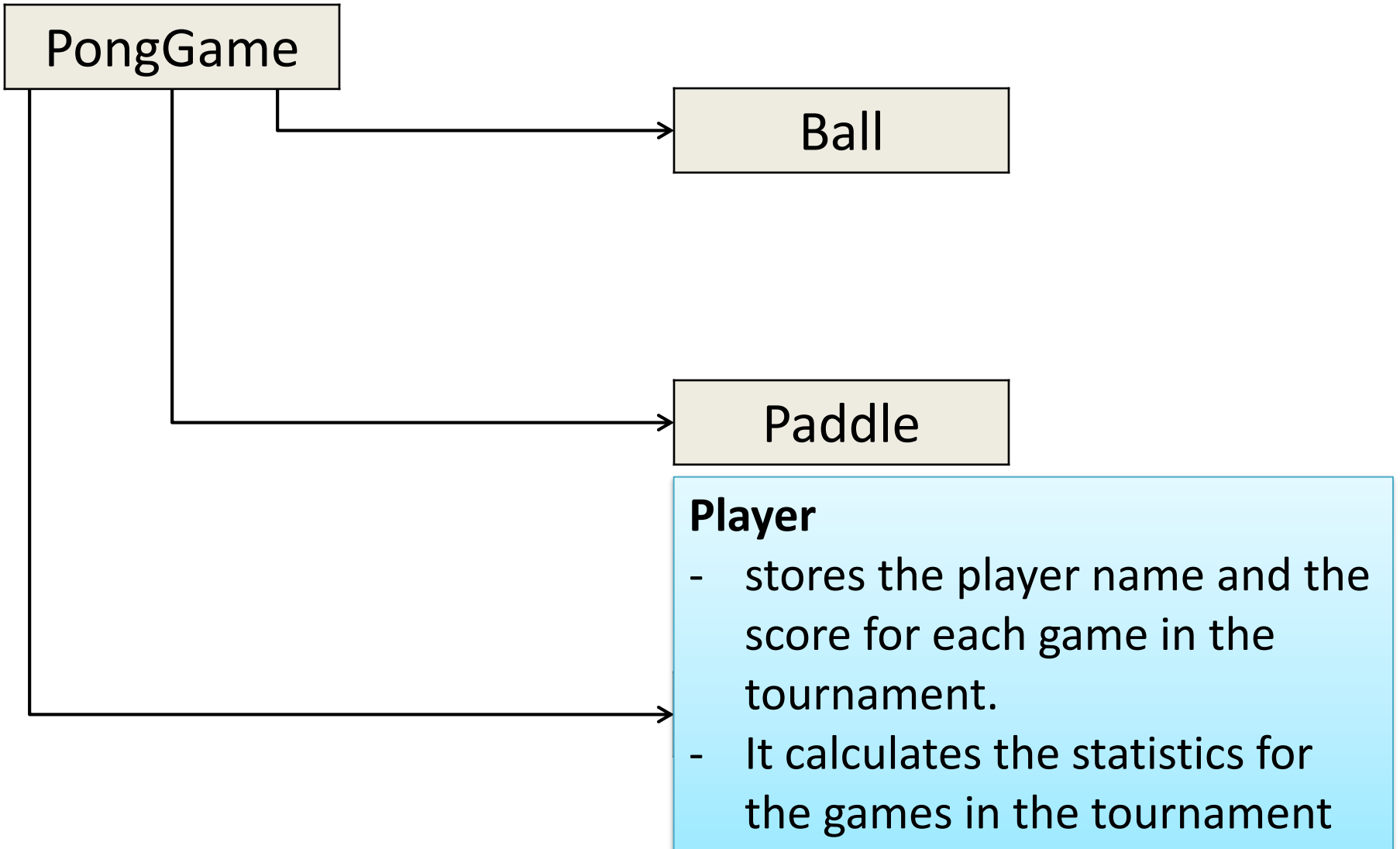
PongGame - Overview



PongGame - Overview

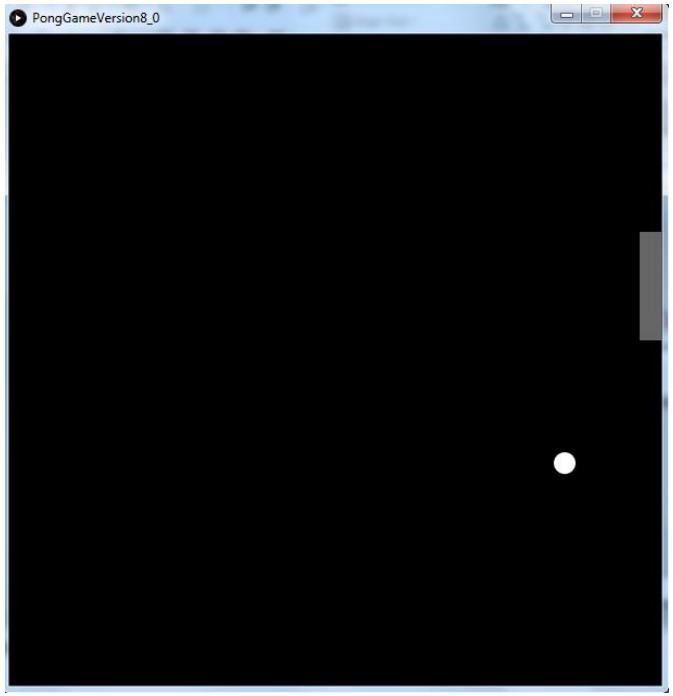


PongGame - Overview



Topics list - **PONG**

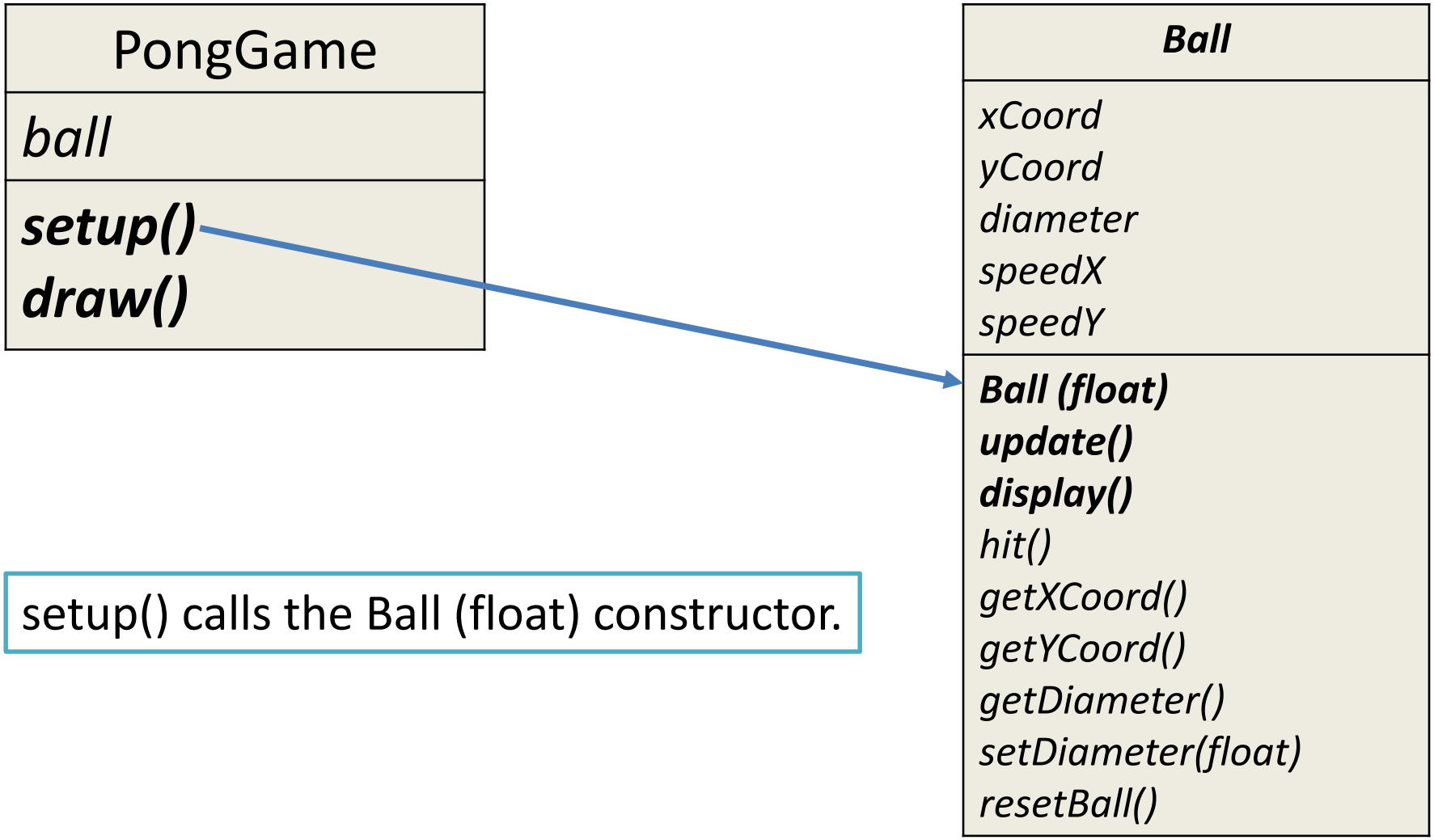
- **Overview** of **PongGame**
- Developing PongGame
  - 9 versions (iterations) described with 4 sets of slides:
  - Set 1
    - V1.0 **(Ball class)**
    - V2.0 (Paddle class)
  - Set 2
    - V3.0 (Collision detection)
    - V4.0 (Lives lost, lives per game, score)
    - V5.0 (Tournament functionality)
  - Set 3
    - V6.0 (Player class – array, no statistics)
    - V7.0 (Player class – array, with statistics)
    - V8.0 (JOptionPane for I/O)
  - Set 4
    - V9.0 (Advanced Collision Detection)



Idea is based on Reas and Fry (2014) example

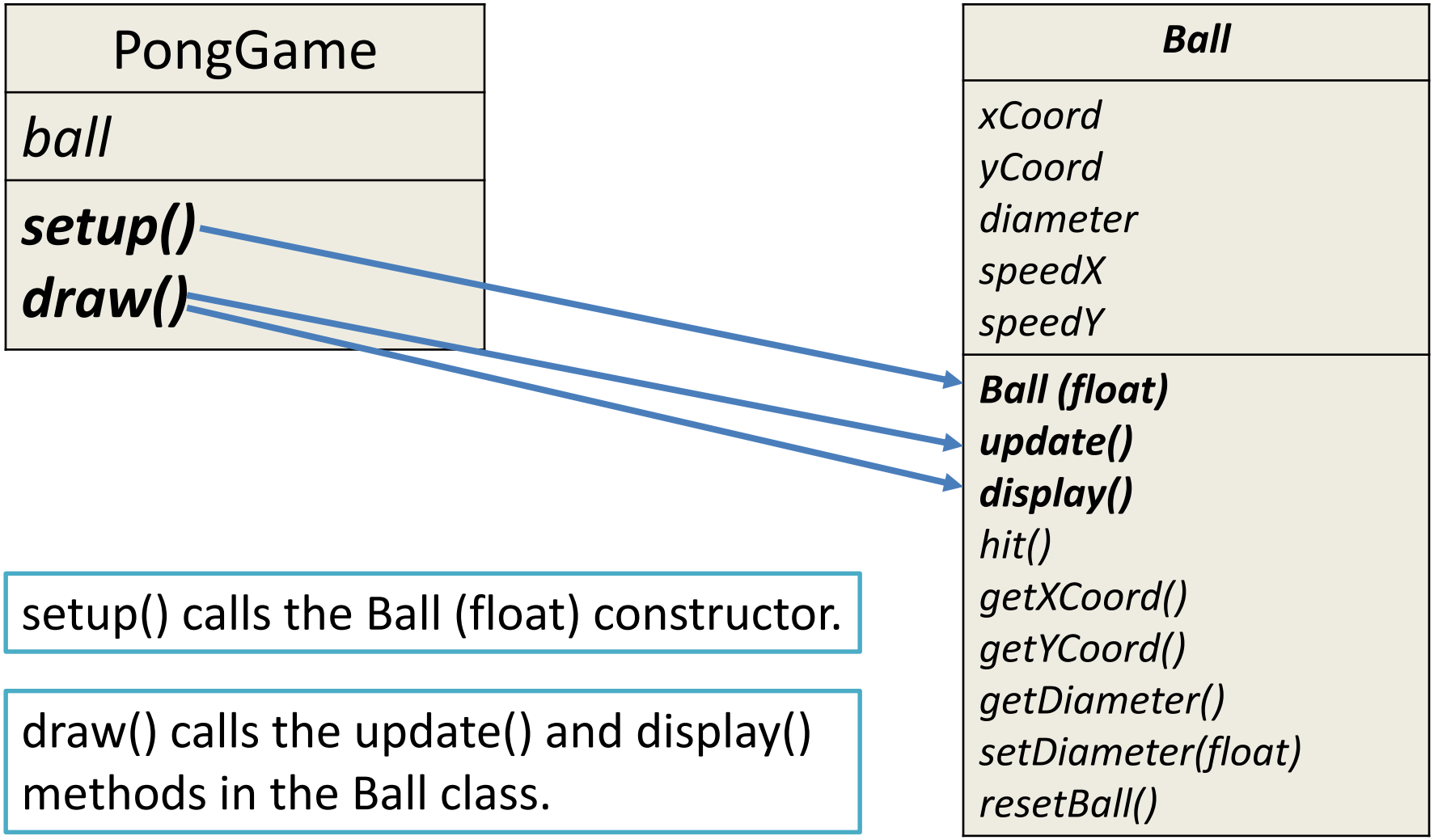
Demo of
Pong Game V1.0

Classes in the PongGameV1.0



setup() calls the Ball (float) constructor.

Classes in the PongGameV1.0



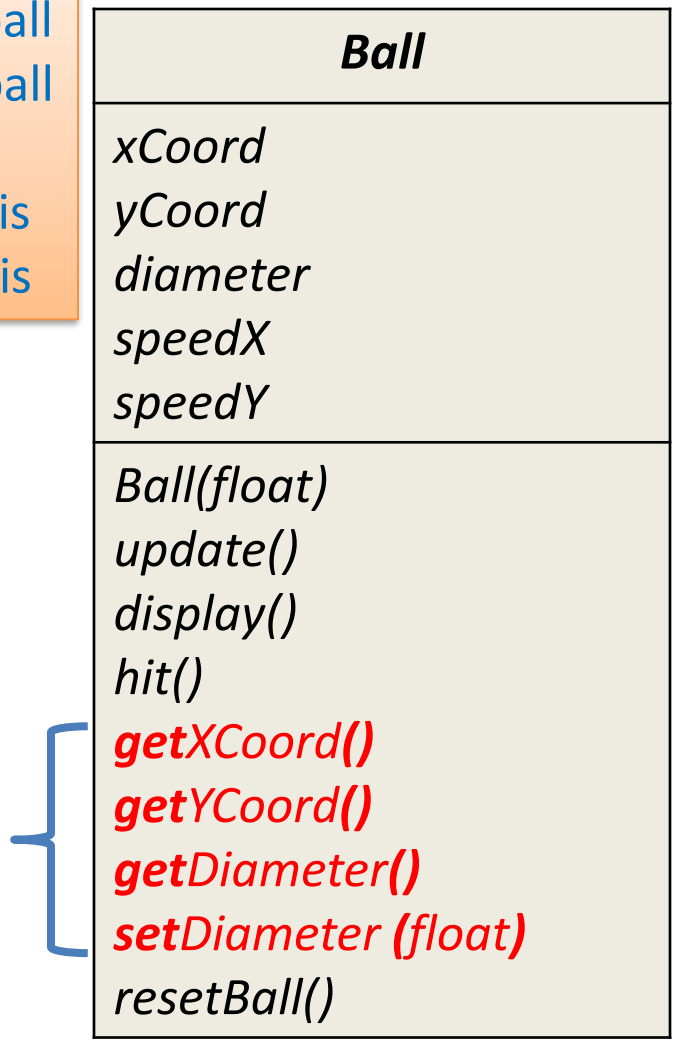
setup() calls the Ball (float) constructor.

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

Ball Class – instance **fields**

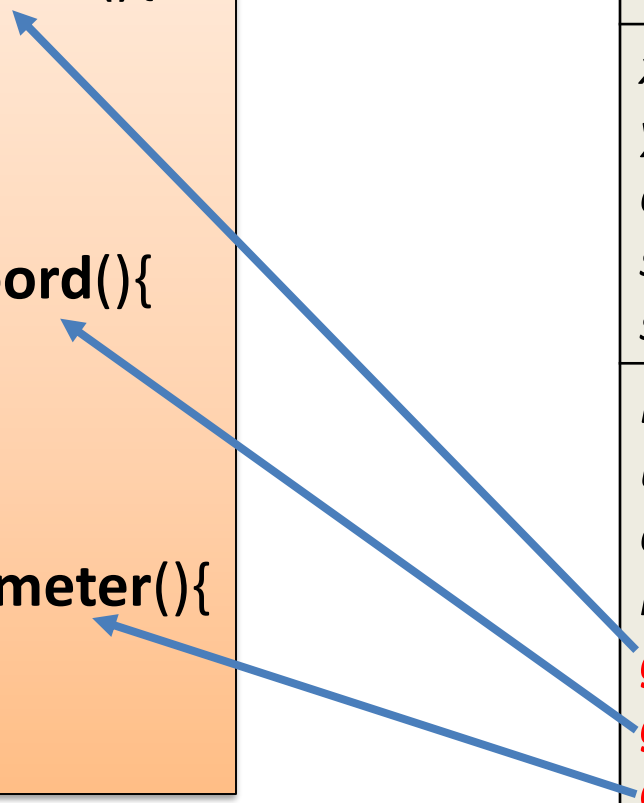
```
private float xCoord;      //x coordinate of the ball
private float yCoord;      //y coordinate of the ball
private float diameter;    //diameter of the ball
private float speedX;      //speed along the x-axis
private float speedY;      //speed along the y-axis
```

getters and **setters** for the **fields**



Ball Class – **get**ters

```
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()* ***get**XCoord**()*** ***get**YCoord**()*** ***get**Diameter**()*** ***set**Diameter **(**float**)*** *resetBall()*

Ball Class – **set**ter

```
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 setter) level;
     // this will depend on your design.
      this.diameter = 20;
   }
}
```

VALIDATION

INITIALISATION

Ball Class – **display()** method

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

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

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

private helper method – resetBall()

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

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

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

private helper method

→ **private** to the class you are in

i.e. can't use it outside of the current class.

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 Class – Ball constructor

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

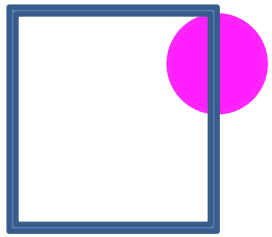
Constructor takes in the diameter of the ball and uses the **setDiameter** *setter method* to update the diameter instance field.

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

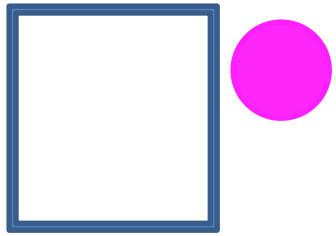
Ball
xCoord *yCoord* *diameter* *speedX* *speedY*
Ball (float) *update()* *display()* *hit()* *getXCoord()* *getYCoord()* *getDiameter()* ***setDiameter** (float)* ***resetBall** ()*

Recap – Drawing Modes: **ellipse**

- The default ellipse mode is CENTER
  - This means x & y positions for ellipse() specify the **center** of the ellipse
  - At the max width of the window, half the ellipse is seen



  - If we specify an x value > width + radius of the circle the circle has left the screen



update() method

update() changes the ball position.

if the ball…

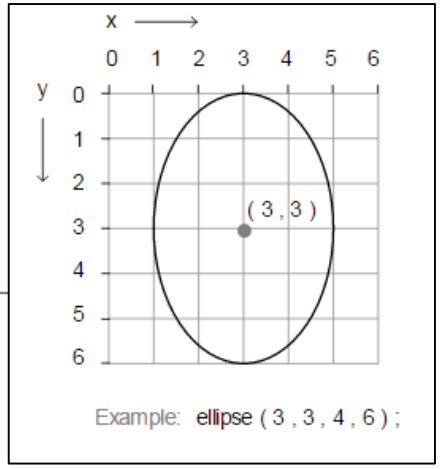
goes **off the screen**
return *true* (i.e. a life was lost)

hits the **left edge**
Change **xCoord** direction

hits the **top or bottom**
Change **yCoord** direction

```
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;
}
```

update() – explained 1

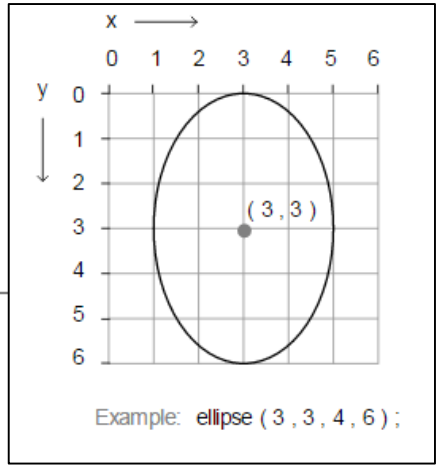



```
//reset position if ball leaves the screen
if (xCoord > width + diameter/2){
   resetBall();
   lifeLost = true;
}
```

(width + diameter/2)

In this check, we add diameter/2 *(i.e. the radius)*
onto the width of the window
so that the ball is completely off the screen
because the x,y values specify the CENTER of the circle

update() – explained 2

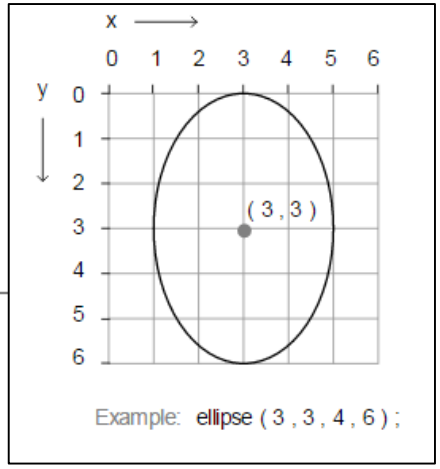



```
// If ball hits the left edge of the display
// window, change direction of xCoord
if (xCoord < diameter/2)
   xCoord = diameter/2;
   speedX = speedX * -1;
}
```

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() – explained 3




```
// 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;
}
```

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

(yCoord < diameter/2)

(yCoord > height - diameter/2)

hit() method

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

We're not using this method
in this version of Pong.

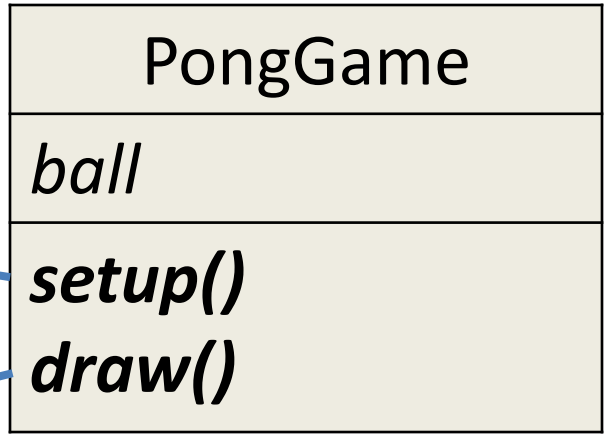
We're preparing our class for
collision detection in V3.0.

This method **changes the ball direction**
when it hits the paddle.
It **bumps 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()*

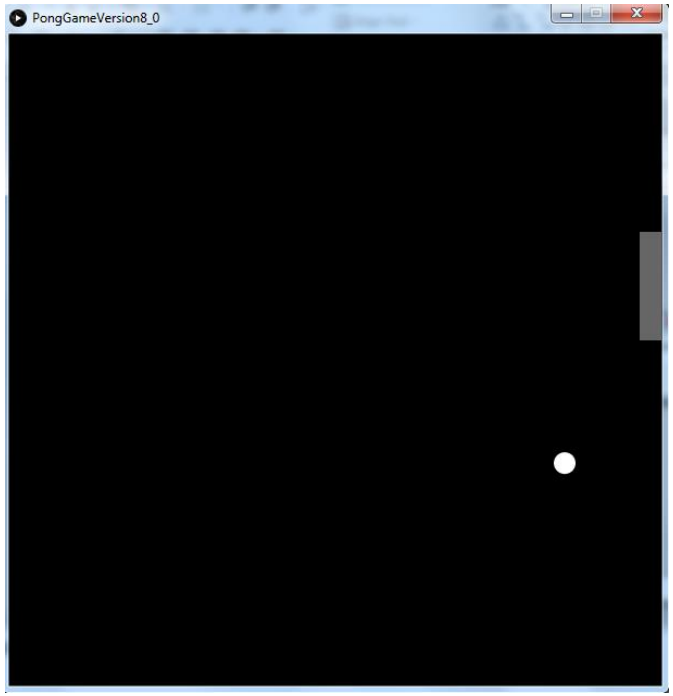
PongGame **V1.0**

```
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();
}
```



Topics list - **PONG**

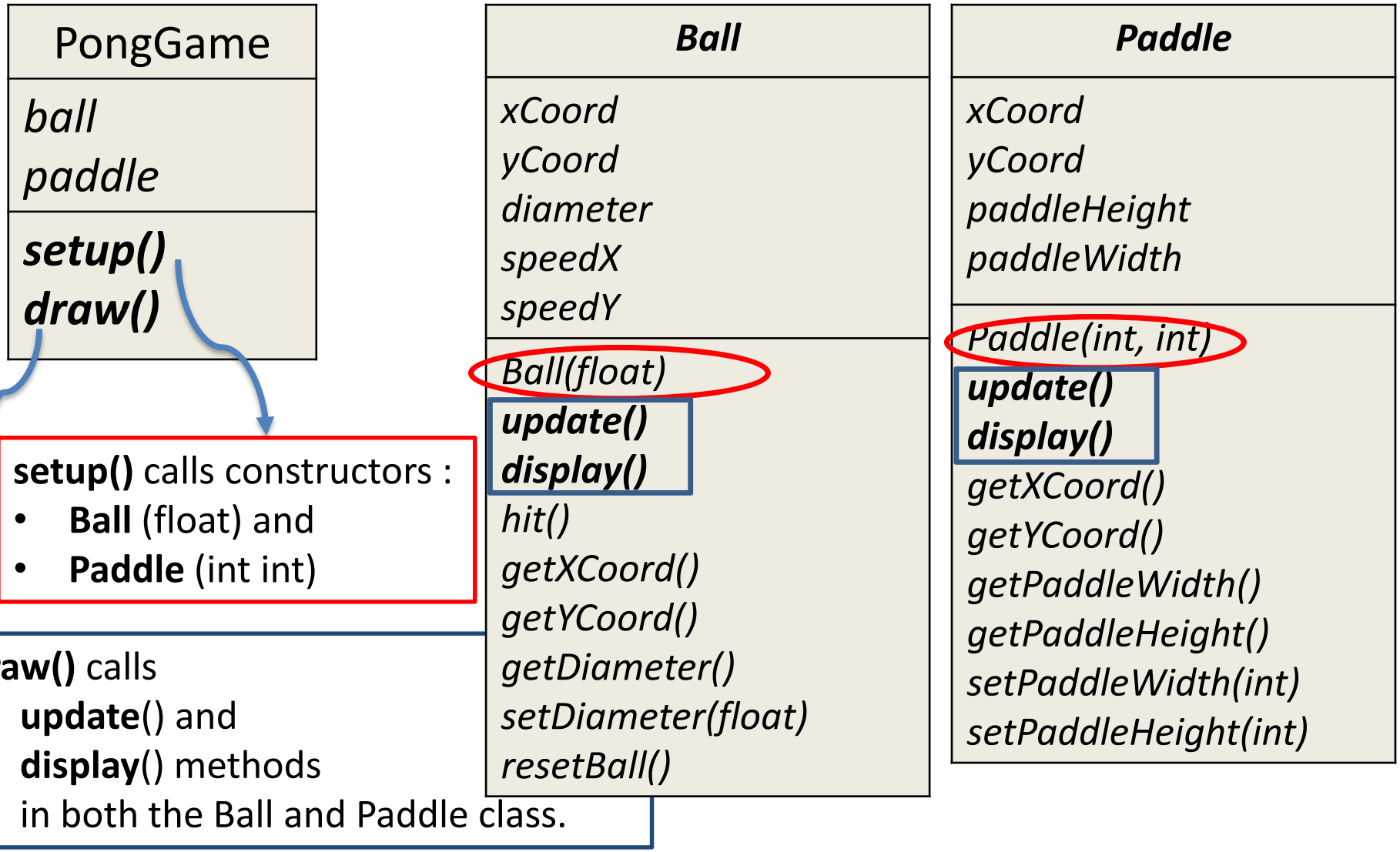
- **Overview** of **PongGame**
- Developing PongGame
  - 9 versions (iterations) described with 4 sets of slides:
  - Set 1
    - V1.0 (Ball class)
    - V2.0 (Paddle class)
  - Set 2
    - V3.0 (Collision detection)
    - V4.0 (Lives lost, lives per game, score)
    - V5.0 (Tournament functionality)
  - Set 3
    - V6.0 (Player class – array, no statistics)
    - V7.0 (Player class – array, with statistics)
    - V8.0 (JOptionPane for I/O)
  - Set 4
    - V9.0 (Advanced Collision Detection)



Idea is based on Reas and Fry (2014) example

Demo of
Pong Game V2.0

Classes in the PongGameV2.0

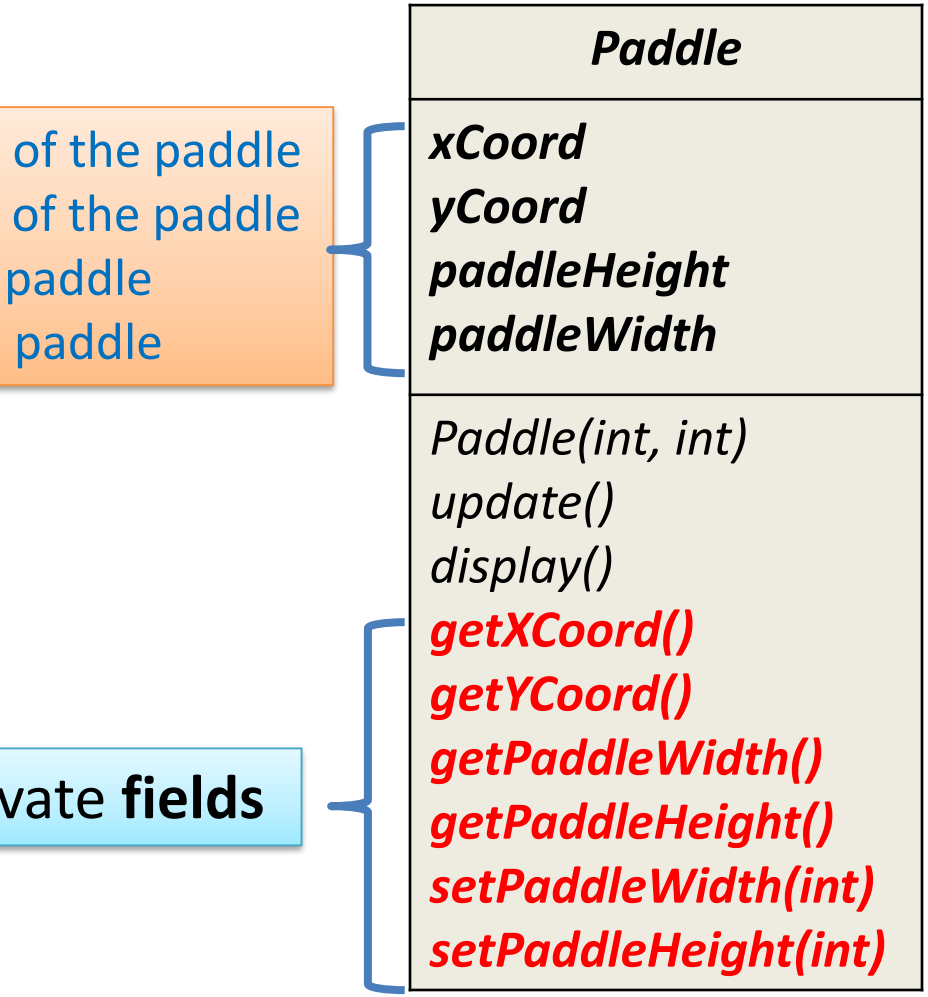


Paddle Class – instance **fields**

```
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
```

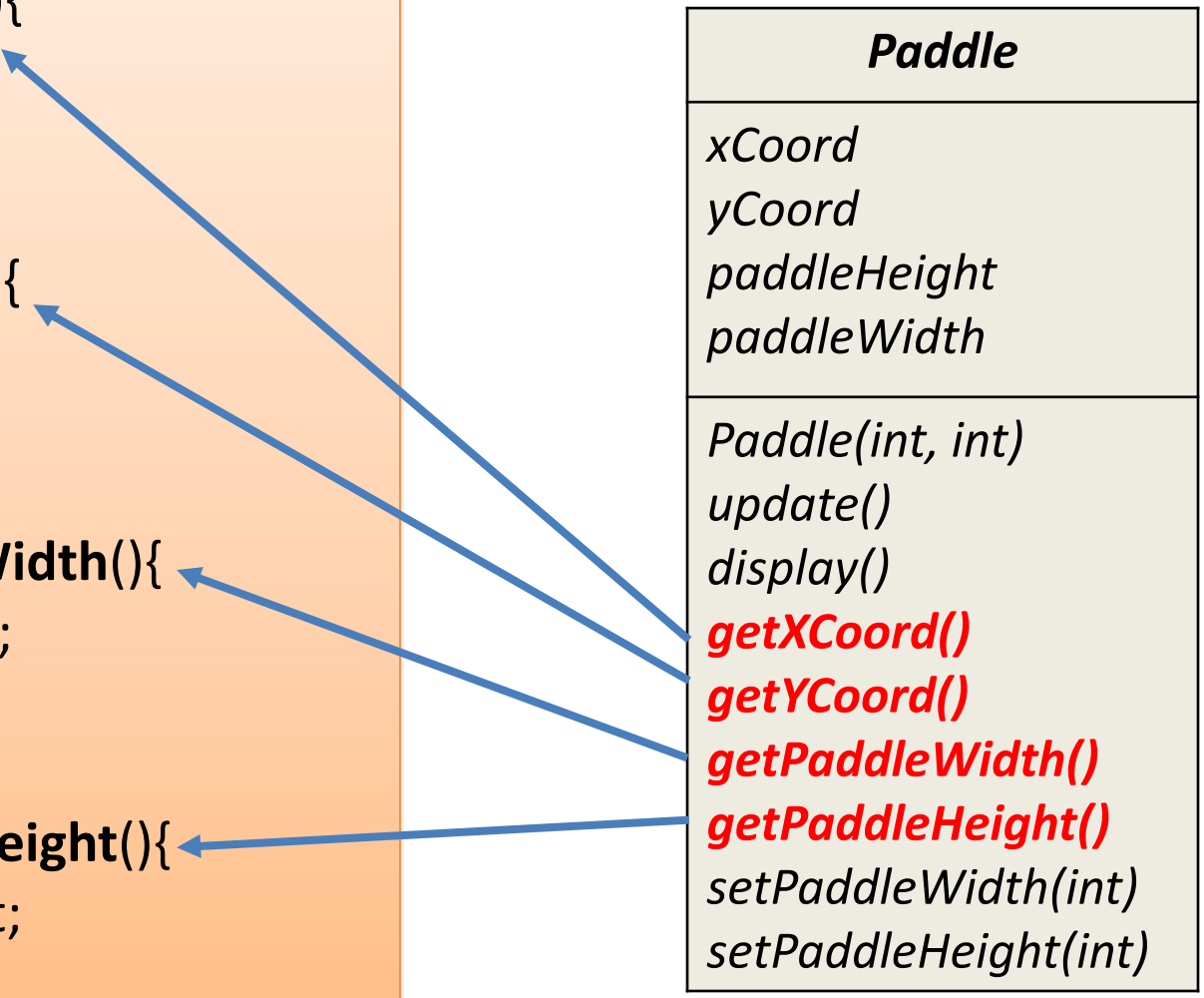
Fields – made private

getters and **set**ters for the private **fields**



Paddle Class – **get**ters

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

public int getYCoord(){
   return yCoord;
}

public int getPaddleWidth(){
   return paddleWidth;
}

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



Paddle Class – **set**ters

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 – **set**ters

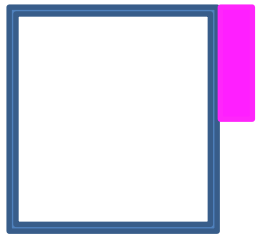
setPaddleHeight(int)

```
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;
    }
  }
```

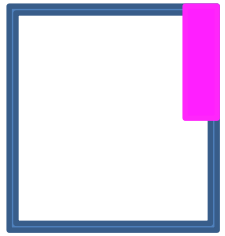
Paddle
xCoord *yCoord* *paddleHeight* *paddleWidth*
Paddle(int, int) *update()* *display()* *getXCoord()* *getYCoord()* *getPaddleWidth()* *getPaddleHeight()* *setPaddleWidth(int)* ***setPaddleHeight(int)***

Recap – Drawing Modes: **rect**

- The default rect mode is CORNER
  - This means x & y positions for rect() specify the **top left CORNER** of the rectangle
  - At the max width of the window, the rectangle would be invisible



  - If we specify an x value which is the width of the screen – width of the rectangle it will be seen



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)*

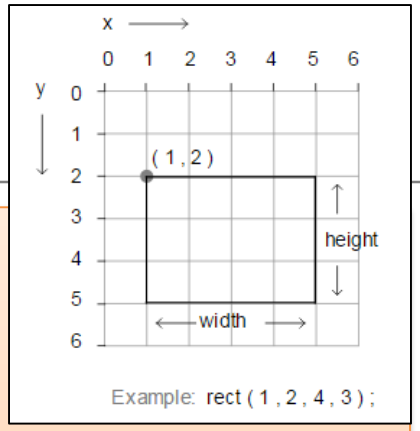
display() method

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

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

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

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;
    }
}
```

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()***

Create Ball & Paddle objects.

Call their update() & display() methods in draw()

Questions?

References

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