

Game of Pong

Developing the game further, V6, V7 and V8.

Produced Dr. Siobhán Drohan
by: Mr. Colm Dunphy
 Mr. Diarmuid O'Connor

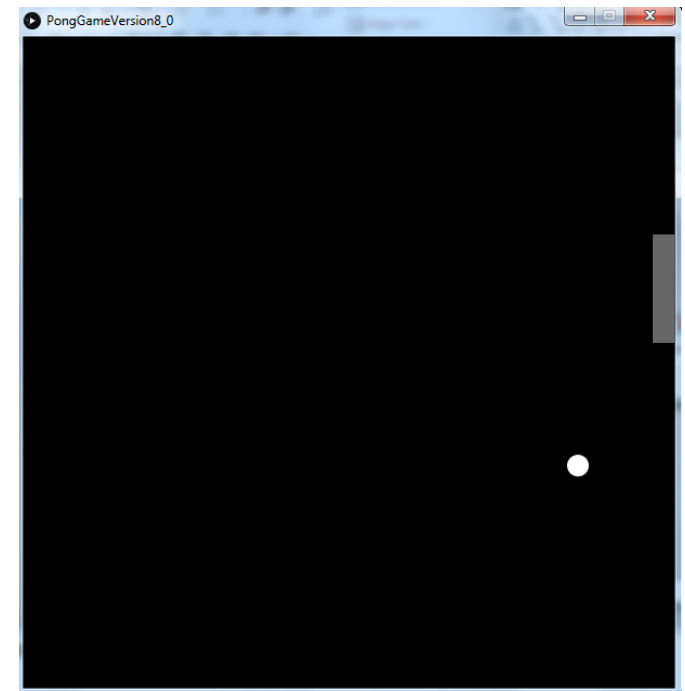


Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

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 (JOptionPane for I/O)



Idea is based on Reas and Fry (2014) example

Demo of Pong Game V6.0

Classes in the PongGameV6.0

PongGame
<i>ball</i> <i>paddle</i> <i>player</i> <i>livesLost</i> score maxLivesPerGame maxNumberOfGames numberOfGamesPlayed
<i>setup()</i> <i>draw()</i> <i>resetGame()</i> <i>tournamentOver()</i> <i>hitPaddle(paddle, ball)</i>

Paddle
<i>Xcoord</i> <i>yCoord</i> <i>paddleHeight</i> <i>paddleWidth</i>
<i>Paddle(int, int)</i> <i>update()</i> <i>display()</i> <i>getXCoord()</i> <i>getYCoord()</i> <i>getPaddleWidth()</i> <i>getPaddleHeight()</i> <i>setPaddleWidth(int)</i> <i>setPaddleHeight(int)</i>

Ball
<i>xCoord</i> <i>yCoord</i> <i>diameter</i> <i>speedX</i> <i>speedY</i>
<i>Ball(float)</i> <i>update()</i> <i>display()</i> <i>hit()</i> <i>getXCoord()</i> <i>getYCoord()</i> <i>getDiameter()</i> <i>setDiameter(float)</i> <i>resetBall()</i>

Player
<i>playerName</i> <i>scores</i> <i>count</i>
<i>addScore(int)</i> <i>getPlayerName()</i> <i>getScores()</i> <i>getCount()</i> <i>setPlayerName(String)</i> <i>setScores(int[])</i> <i>toString()</i>

We have a new **Player** class.

This stores the **score** of the current player in an array

Use of Arrays in Player

- We use an array of integers
 - to hold the **scores** for the games.
- declare at start:
`private int[] scores;`
- and in constructor:
`scores = new int[numOfGames]`
- The **addScore()** method
 - adds a score to this array when called (by PongGame).

Player class

```
private String playerName;
```

```
private int[] scores;           //array to hold scores
```

```
private int count;              //holds current position  
                                //in array
```

private fields

getters and setters for fields above

Player

playerName

scores

count

addScore(int)

getPlayerName()

getScores()

getCount()

setPlayerName(String)

setScores(int[])

toString()

Player class – addScore() method

```
public void addScore (int score) {  
    if (score >= 0){  
        scores[count] = score;  
        count++;  
    }  
}
```

addScore (int)

- takes in the new score as a parameter.
- adds the new score to the array
- increments the counts variable

scores

0	4
1	5
2	
3	

+

Before

score

3

After

0	4
1	5
2	3
3	

count

3

count

2

Player

playerName

scores

count

addScore (int)

getPlayerName()

getScores()

getCount()

setPlayerName(String)

setScores(int[])

toString()

Player class – toString() method

```
public String toString () {  
    String str = "Scores for " + playerName + "\n";  
    for(int i = 0; i < count; i++){  
        str = str + "    Score " + (i+1) + ": " +  
            scores[i] + "\n";  
    }  
    return str;  
}
```

toString()

- **returns a string version of an object.**
- useful method
 - we will have a toString() method in most classes.

scores

0	4
1	5
2	3
3	4

count

4

toString()
returns

"Score 1 : 4 \n"+
"Score 2 : 5 \n"+
"Score 3 : 3 \n" +
"Score 4 : 4\n"

Player

playerName
scores
count

addScore
getPlayerName()
getScores()
getCount()
setPlayerName(String)
setScores(int[])
toString()

When is the **Player** object used?

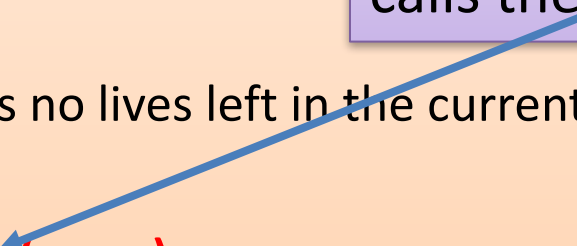
```
Ball ball;  
Paddle paddle;  
Player player;  
:  
void setup(){  
    size(600,600);  
    noCursor();  
    //setting up ball and paddle with hard-coded sizes.  
    ball = new Ball(20.0);  
    paddle = new Paddle(20,100);  
    //create a player object  
    player = new Player(" PongMaster ", maxNumberOfGames);  
}
```

Need to declare
and setup Player

When is the **Player** object used?

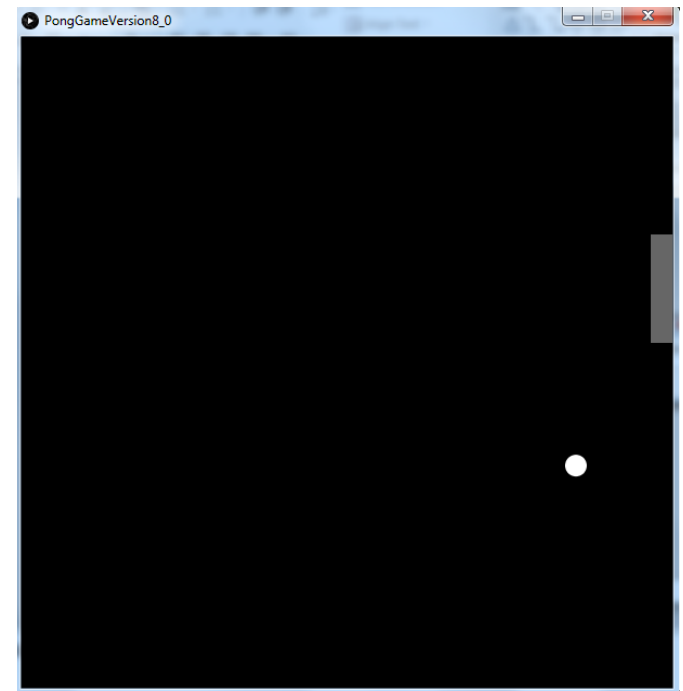
```
void draw(){  
  background(0);  
  paddle.update();  
  :  
  //If the player has no lives left in the current game  
  else{  
    player.addScore(score);  
    numberOfGamesPlayed++;  
    if (numberOfGamesPlayed < maxNumberOfGames){  
      resetGame();  
    }  
    else  
      tournamentOver();  
  }  
}
```

'Sends a message to the player object to add a new score to its scores array.'
i.e.
calls the **addScores()** method.



Topics list - PONG

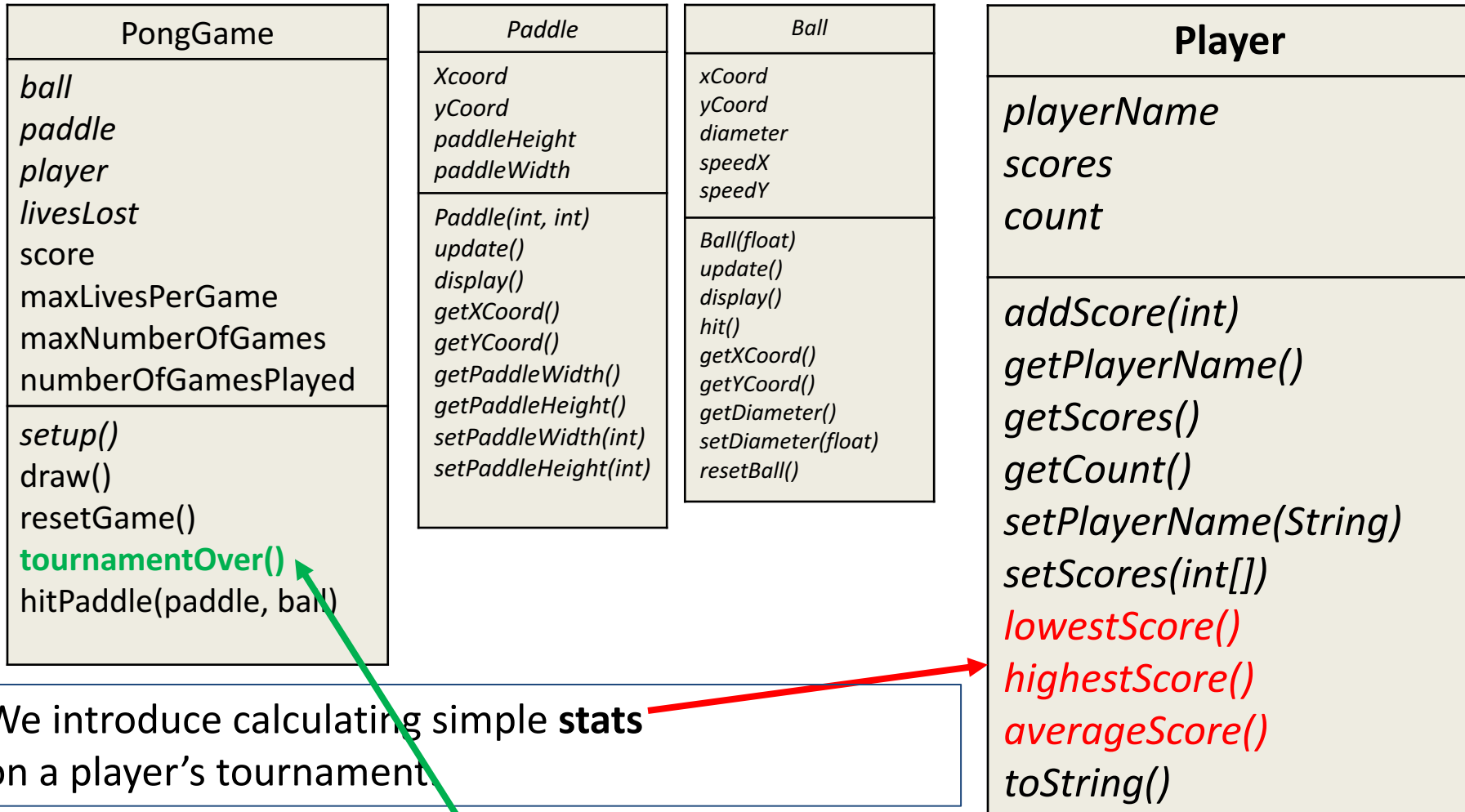
- Overview of PongGame
- Developing PongGame
 - 9 versions (iterations) described with 4 sets of slides:
 - Set 1
 - V1.0 (Ball class)
 - V2.0 (Paddle class)
 - Set 2
 - V3.0 (**Collision detection**)
 - V4.0 (Lives lost, lives per game, score)
 - V5.0 (Tournament functionality)
 - Set 3
 - V6.0 (Player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



Idea is based on Reas and Fry (2014) example

Demo of Pong Game V7.0

Classes in the PongGameV7.0



Methods to calculate statistics

- When the players tournament is over, we calculate the player's
 - **highest** score
 - **lowest** score.
 - **average** score.
- Values are calculated within the **Player** class
 - as we have enough data there to do this (scores array).
- These methods are then called from the **tournamentOver()** method in the PongGame class.

highestScore()

```
public int highestScore () {  
  
    int highestScore = scores[0];  
  
    for(int i = 1; i < count; i++){  
        if (scores[i] > highestScore){  
            highestScore = scores[i];  
        }  
    }  
    return highestScore;  
}
```

We use a variable (**highestScore**) to store the highest score we have seen in the scores array so far.

If the next value in the array is larger than this highest so far value, then we make the highest value equal this new highest value.

Player
<i>playerName</i> <i>scores</i> <i>count</i>
<i>addScore</i> <i>getPlayerName()</i> <i>getScores()</i> <i>getCount()</i> <i>setPlayerName(String)</i> <i>setScores(int[])</i> <i>lowestScore()</i> <i>highestScore()</i> <i>averageScore()</i> <i>toString()</i>

lowestScore()

```
public int lowestScore() {  
    int lowestScore = scores[0];  
  
    for(int i = 1; i < count; i++){  
        if (scores[i] < lowestScore){  
            lowestScore = scores[i];  
        }  
    }  
    return lowestScore;  
}
```

We use a variable (**lowestScore**) to store the lowest score we have seen in the scores array so far.

If the next value in the array is smaller than this lowest so far value, then we make the lowest value equal this new lowest value.

Player
<i>playerName</i> <i>scores</i> <i>count</i>
<i>addScore</i> <i>getPlayerName()</i> <i>getScores()</i> <i>getCount()</i> <i>setPlayerName(String)</i> <i>setScores(int[])</i> <i>lowestScore()</i> <i>highestScore()</i> <i>averageScore()</i> <i>toString()</i>

averageScore()

```
public int averageScore() {  
  
    int total = 0;  
  
    for(int i = 0; i < count; i++){  
        total = total + scores[i];  
    }  
  
    return total / count;  
}
```

Player
<i>playerName</i> <i>scores</i> <i>count</i>
<i>addScore</i> <i>getPlayerName()</i> <i>getScores()</i> <i>getCount()</i> <i>setPlayerName(String)</i> <i>setScores(int[])</i> <i>lowestScore()</i> <i>highestScore()</i> <i>averageScore()</i> <i>toString()</i>

We total up all the scores and get the average by dividing the sum by the number of values (in count).

Where the **stats methods** are used...

```
void tournamentOver(){  
    println ("Game Over!\n");  
    println (player.getPlayerName()  
        + ", your tournament is over!\n"  
        + "Number of games played: "  
        + numberOfGamesPlayed  
        + "\n\n"  
        + player.toString()  
        + "\n\nHighest Score: " + player.highestScore()  
        + "\nLowest Score: " + player.lowestScore()  
        + "\nAverage Score: " + player.averageScore());  
    exit();  
}
```

This method calls the **stats methods** on the player object:

player.highestScore
player.lowestScore
player.averageScore

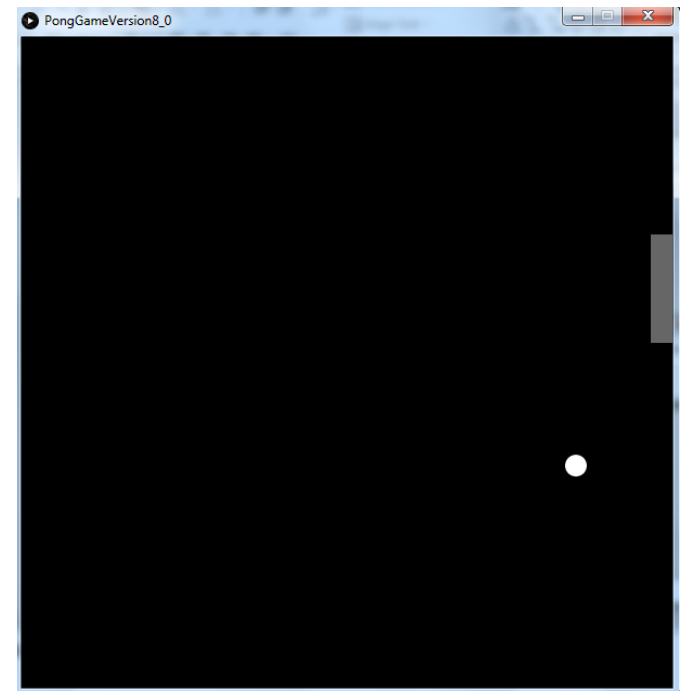
PongGame
<i>ball</i> <i>paddle</i> <i>player</i> <i>livesLost</i> <i>score</i> <i>maxLivesPerGame</i> <i>maxNumberOfGames</i> <i>numberOfGamesPlayed</i>
<i>setup()</i> <i>draw()</i> <i>resetGame()</i> tournamentOver() <i>hitPaddle(paddle, ball)</i>

A few things to note

- We did not need to change any methods in Paddle or Ball during this version update.
- The changes to Player and PongGame methods did not effect the other methods already written.

Topics list - PONG

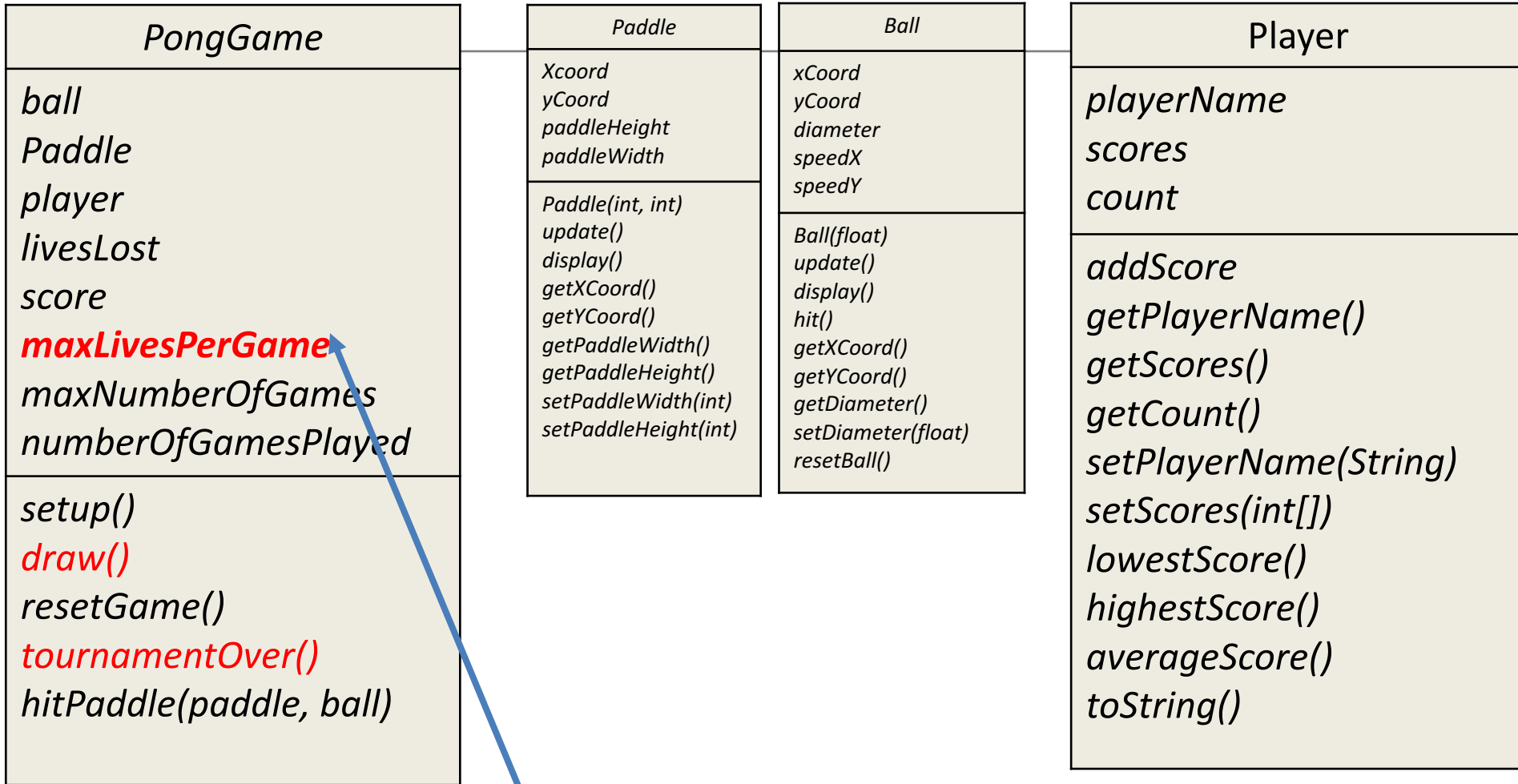
- Overview of PongGame
- Developing PongGame
 - 9 versions (iterations) described with 4 sets of slides:
 - Set 1
 - V1.0 (Ball class)
 - V2.0 (Paddle class)
 - Set 2
 - V3.0 (**Collision detection**)
 - V4.0 (Lives lost, lives per game, score)
 - V5.0 (Tournament functionality)
 - Set 3
 - V6.0 (Player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



Idea is based on Reas and Fry (2014) example

Demo of Pong Game V8.0

Classes in the PongGameV8.0



JOptionPane allows user input, during the running of the program.


We use this input to make changes in the game.

A few things to note

- We only use data input or data output in the `PongGame(Driver)` class.
- This is to ensure that the ‘user of classes’ (`PongGame`) gets to decide how the data is input and output.
- This is why **`toString()`** is useful
 - it returns a string version of an object of a class
 - then the user can decide how to show it
e.g. on the console or via `JOptionPane`.

import JOptionPane

In order to use **JOptionPane**, we must **import swing** at the top of the file.



```
import javax.swing.*;
```

```
//Objects required in the program
```

```
Ball ball;
```

```
Paddle paddle;
```

```
Player player;
```

```
:
```

PongGame

ball

paddle

player

livesLost

score

maxLivesPerGame

maxNumberOfGames

numberOfGamesPlayed

setup()

draw()

resetGame()

tournamentOver()

hitPaddle(paddle, ball)

Reading in maxNumberOfGames

```
int maxNumberOfGames;  
//code omitted  
maxNumberOfGames =  
    Integer.parseInt (JOptionPane.showInputDialog(  
        "Welcome to the Pong Tournament\n\n  
        Please enter the number of games you would like to play:",  
        "3"));  
  
player = new Player (JOptionPane.showInputDialog(  
    "Enter the player name (max 6 chars: ")", maxNumberOfGames);
```

PongGame

ball
paddle
player
livesLost
score
maxLivesPerGame
maxNumberOfGames
numberOfGamesPlayed

setup()
draw()
resetGame()
tournamentOver()
hitPaddle(paddle, ball)

Reading in maxNumberOfGames

maxNumberOfGames is read in

```
int maxNumberOfGames;  
//code omitted  
maxNumberOfGames =  
    Integer.parseInt(JOptionPane.showInputDialog(  
        "Welcome to the Pong Tournament\n\n  
        Please enter the number of games you would like to play:",  
        "3"));  
player = new Player (JOptionPane.showInputDialog("Enter the player  
        name (max 6 chars: "), maxNumberOfGames);
```

PongGame

```
ball  
paddle  
player  
livesLost  
score  
maxLivesPerGame  
maxNumberOfGames  
numberOfGamesPlayed  
  
setup()  
draw()  
resetGame()  
tournamentOver()  
hitPaddle(paddle, ball)
```

Reading in maxNumberOfGames

```
int maxNumberOfGames;  
//code omitted  
maxNumberOfGames =  
    Integer.parseInt(JOptionPane.showInputDialog(  
        "Welcome to the Pong Tournament\n\n  
        Please enter the number of games you would like to play:",  
        "3"));  
player = new Player (JOptionPane.showInputDialog("Enter the player  
        name (max 6 chars: "), maxNumberOfGames);
```

The Player constructor is called and the JOptionPane input is passed into the constructor.

PongGame
<i>ball</i> <i>paddle</i> player <i>livesLost</i> <i>score</i> <i>maxLivesPerGame</i> maxNumberOfGames <i>numberOfGamesPlayed</i>
setup() <i>draw()</i> <i>resetGame()</i> <i>tournamentOver()</i> <i>hitPaddle(paddle, ball)</i>

Adding choice during the game

- Having read in the maximum number of games a player can have,
the **player is asked at the end of each game if they wish to continue.**
 - If they choose to end, their tournament is over.
- When max number of games as read in, is reached
 - they will finish without being asked.

Adding choice during the game

V7

```
//If the player has no lives left in the current game
else{
    //add the score of the current game to the array in player
    player.addScore(score);
    numberOfGamesPlayed++;
    //If the player has more games left in the tournament,
    //display their score and ask them if they want to
    //continue with the tournament.
    if (numberOfGamesPlayed < maxNumberOfGames){
        resetGame();
    }
    else{
        //the player has no more games left in the tournament
        tournamentOver();
    }
}
```

PongGame

ball
paddle
player
livesLost
score
maxLivesPerGame
maxNumberOfGames
numberOfGamesPlayed

setup()
draw()
resetGame()
tournamentOver()
hitPaddle(paddle, ball)

Adding choice during the game

```
//If the player has no lives left in the current game
```

```
else{  
    player.addScore(score);  
    numberOfGamesPlayed++;  
    if (numberOfGamesPlayed < maxNumberOfGames){  
        int reply = JOptionPane.showConfirmDialog(null,  
            "Game Over! You scored " + score +  
            ".\nWould you like to play the next game in your tournament?",  
            "Play next game?", JOptionPane.YES_NO_OPTION);  
        if (reply == JOptionPane.YES_OPTION){  
            resetGame();  
        }  
        else{  
            tournamentOver();  
        }  
    }  
}
```

V8

PongGame

<i>ball</i> <i>paddle</i> <i>player</i> <i>livesLost</i> <i>score</i> <i>maxLivesPerGame</i> <i>maxNumberOfGames</i> <i>numberOfGamesPlayed</i>
<i>setup()</i> <i>draw()</i> <i>resetGame()</i> <i>tournamentOver()</i> <i>hitPaddle(paddle, ball)</i>

We added extra functionality here, based on our new field **maxNumberOfGames** and **JOptionPane**.

JOptionPane for output

```
void tournamentOver ()
{
    JOptionPane.showMessageDialog(null,
        player.getPlayerName() +
        ", your tournament is over! \n\n" +
        "Number of games played: " +
        numberOfGamesPlayed + "\n\n" +
        player.toString() +
        "\n\nHighest Score: " + player.highestScore() +
        "\nLowest Score: " + player.lowestScore() +
        "\nAverage Score: " + player.averageScore());
    exit();
}
```

<i>PongGame</i>
<i>ball</i> <i>paddle</i> <i>player</i> <i>livesLost</i> <i>score</i> <i>maxLivesPerGame</i> <i>maxNumberOfGames</i> <i>numberOfGamesPlayed</i>
<i>setup()</i> <i>draw()</i> <i>resetGame()</i> <i>tournamentOver()</i> <i>hitPaddle(paddle, ball)</i>

The same data is being output, just in a better way...
we are using **JOptionPane** instead of the console.

Questions?



References

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