Game of Pong

Developing the game further

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

- 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

Recap: Pong Game so Far

- Main issues to note
  - Different Classes used for different entities (real-world artefacts)
  - How do we choose where to put methods?
  - The ‘Game’ class brings everything together
- Collision detection algorithm
  - Original (ver 5) is more re-usable but harder to understand
  - Later (ver 8_1) is easier to understand but harder to reuse.

Source: Reas & Fry (2014)

Demo of

Pong Game V6.0

Classes in the PongGameV6.0

PongGame
ball
Paddle
livesLost
score
maxLivesPerGame
maxNumberOfGames
numberOfGamesPlayed
setup()
draw()
resetGame()
tournamentOver()
hitPaddle(paddle, ball)

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

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()

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;
  ```

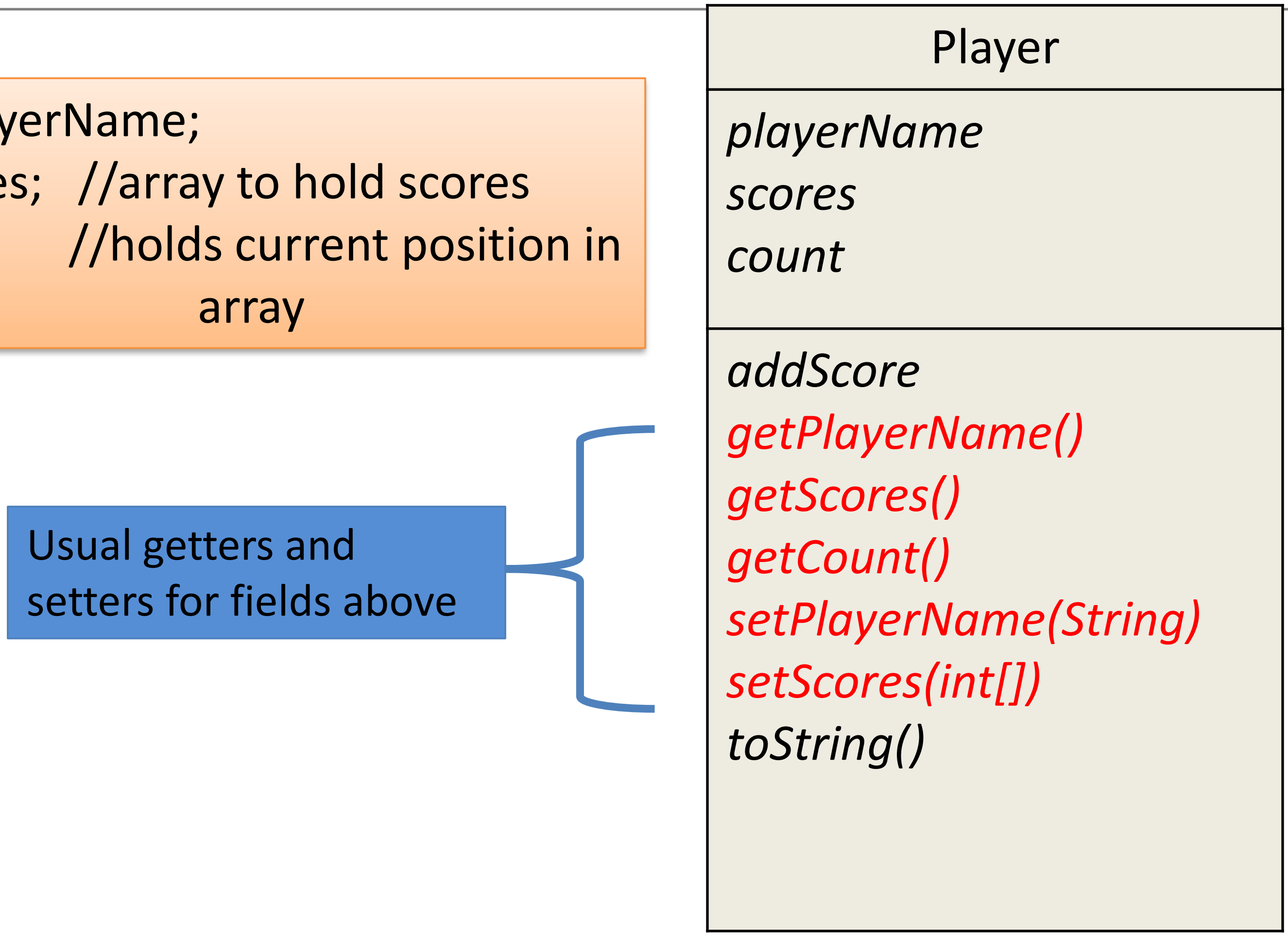
  and in constructor:

  ```
  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
```



Player

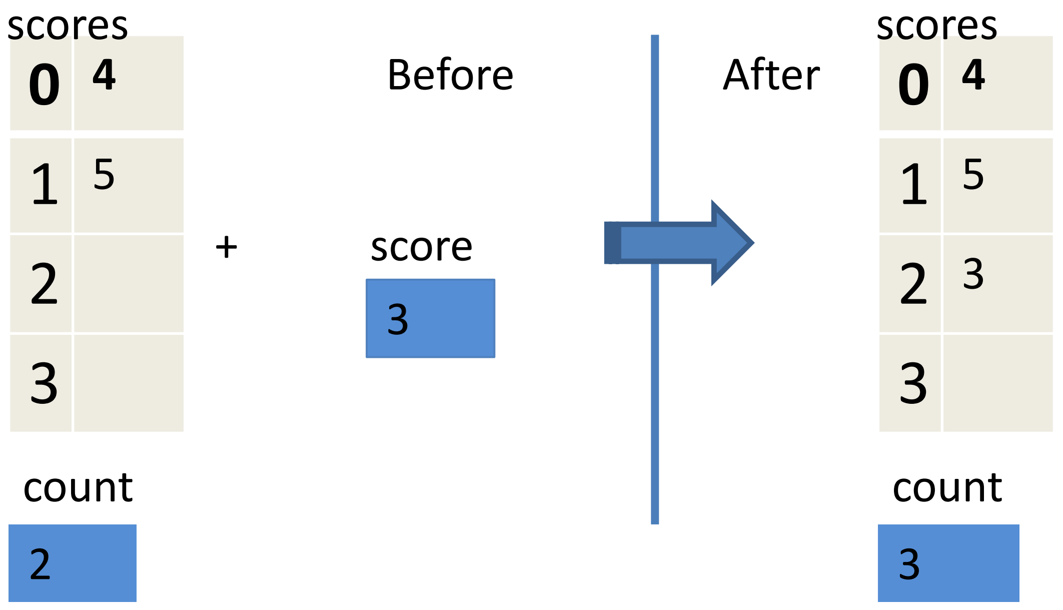
playerName
scores
count

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

Usual getters and setters for fields above

Player class – addScore method

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



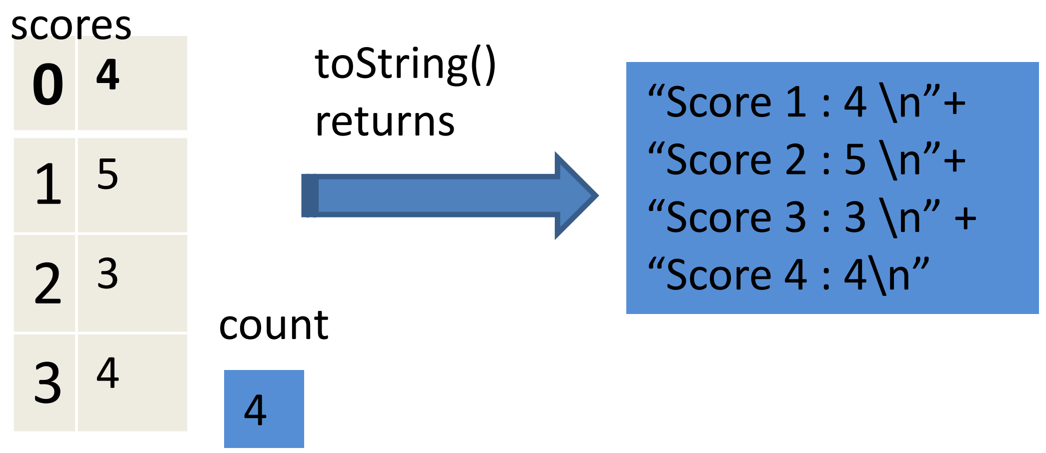
Player
playerName *scores* *count*

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

The addScore method takes in
The new score as a parameter. It
adds the new score to the array
and increments the counts variable

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;
}
```



Player
playerName *scores* *count*

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

The toString() method returns
A string version of an object. This
Is a useful method and we will have
A toString() method in most classes.

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.' or Calls the addScores() method.

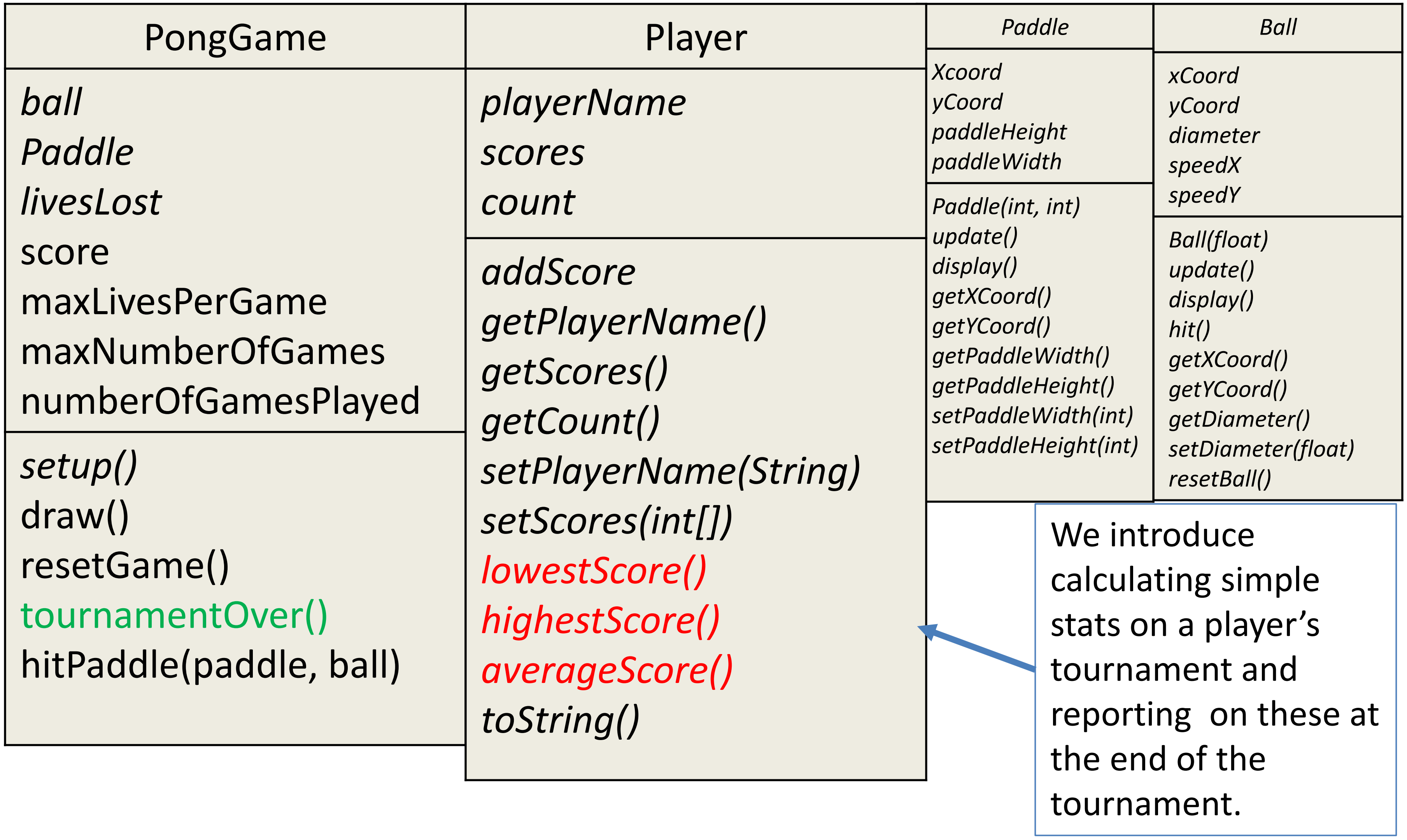
Topics list

- 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 V7.0

Classes in the PongGameV7.0



PongGame	Player	Paddle	Ball
ball *Paddle* *livesLost* score maxLivesPerGame maxNumberOfGames numberOfGamesPlayed	*playerName* *scores* *count*	*Xcoord* *yCoord* *paddleHeight* *paddleWidth*	*xCoord* *yCoord* *diameter* *speedX* *speedY*

setup()
draw()
resetGame()
tournamentOver()
hitPaddle(paddle, ball)
	addScore
	getPlayerName()
	getScores()
	getCount()
	setPlayerName(String)
	setScores(int[])
	lowestScore()
	highestScore()
	averageScore()
	toString()
		Paddle(int, int)
		update()
		display()
		getXCoord()
		getYCoord()
		getPaddleWidth()
		getPaddleHeight()
		setPaddleWidth(int)
		setPaddleHeight(int)
			Ball(float)
			update()
			display()
			hit()
			getXCoord()
			getYCoord()
			getDiameter()
			setDiameter(float)
			resetBall()

We introduce calculating simple stats on a player's tournament and reporting on these at the end of the tournament.

Methods to calculate statistics

- When the players tournament is over, we calculate
  - The player’s highest score
  - The player’s lowest highest score.
  - The player’s average score.

- These values can be calculated within the Player class as we have enough data there to do this (from the scores array).

- These methods are then called from the tournamentOver() method in the PongGame class.

Method 1 – 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.

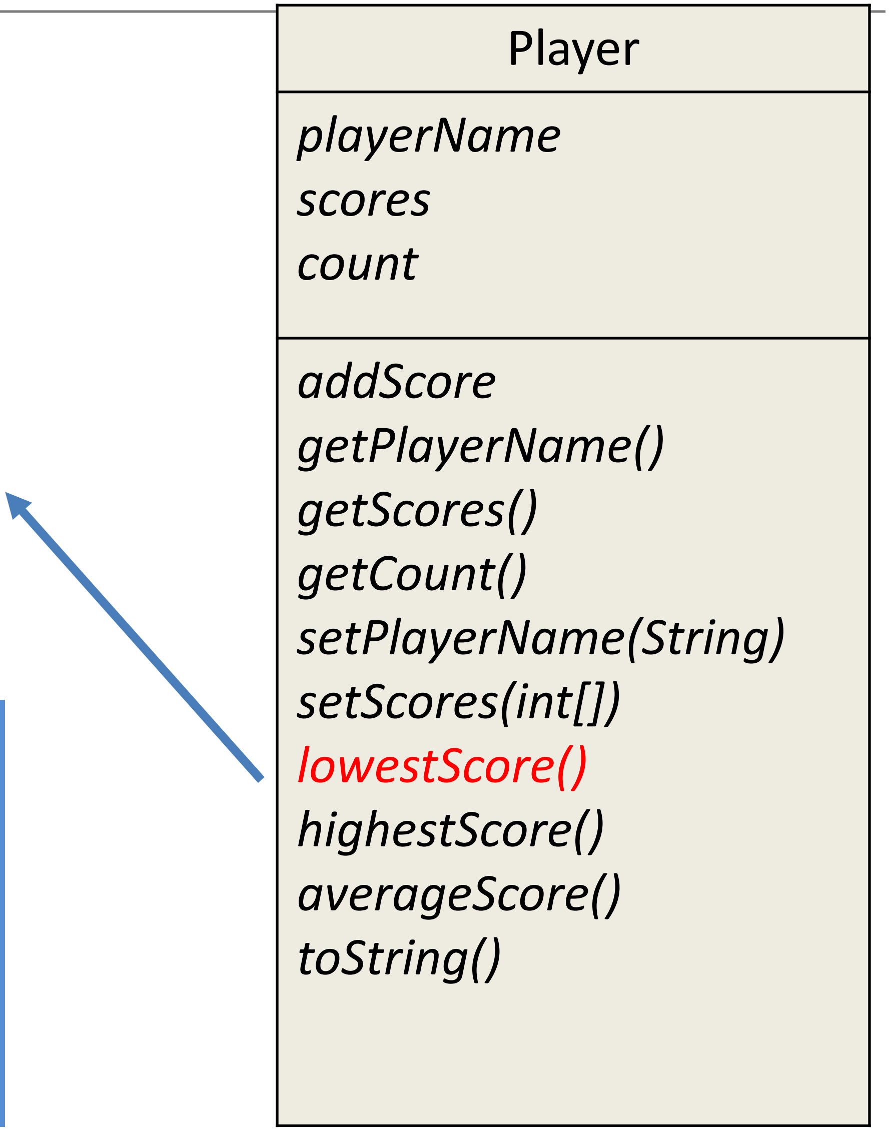
Player
playerName *scores* *count*

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

Method 2 – lowestScore()

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

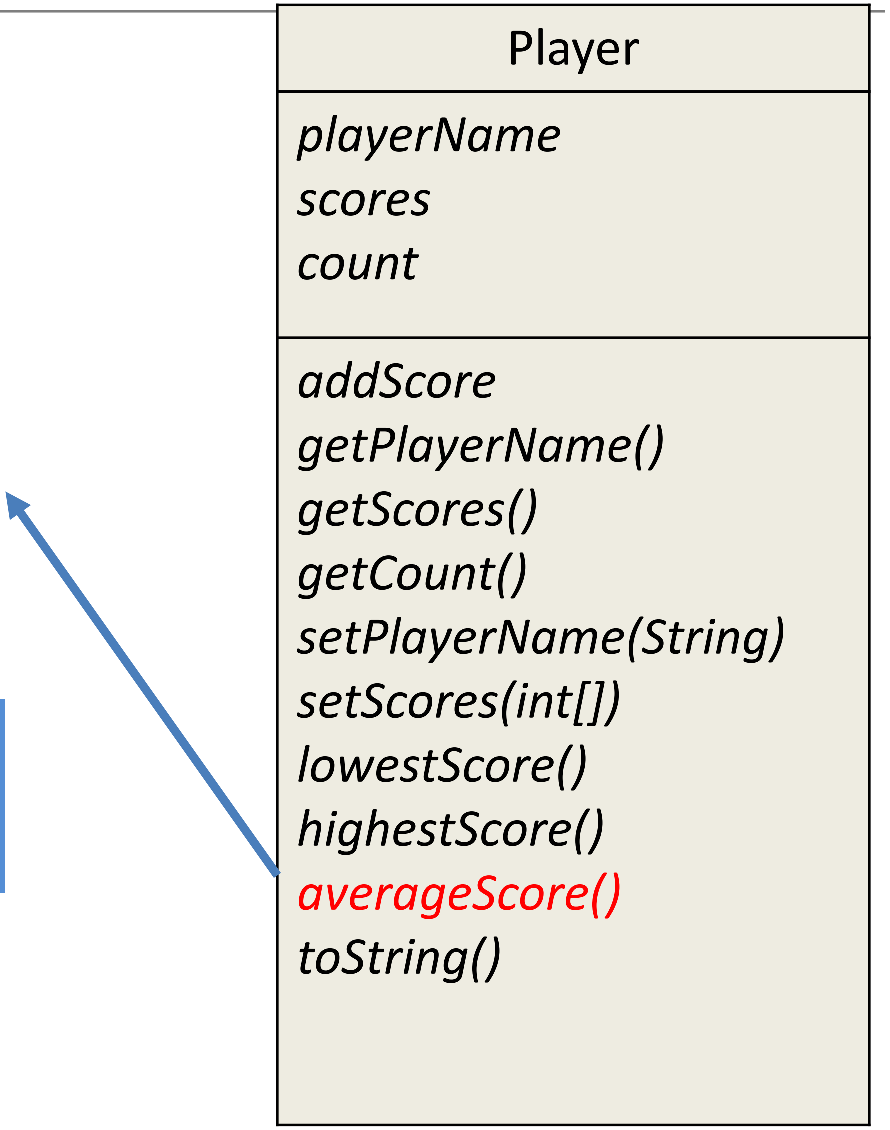
We use a variable (lowestScore) to store the lowest score we have seen in the scores array so far.
If the next value in the array is smaller than this lowest so far value, then we make the lowest value equal this new lowest value.



Method 3 – averageScore()

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

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



Where 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();
 }
```

PongGame
ball *Paddle* *livesLost* score maxLivesPerGame maxNumberOfGames numberOfGamesPlayed

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

This method just calls the stats method on the player object.

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

- 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 V8.0

Classes in the PongGameV8.0

PongGame
ball *Paddle* *livesLost* score maxLivesPerGame maxNumberOfGames numberOfGamesPlayed

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

Player
playerName *scores* *count*

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

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()

We introduce the use of JOptionPane to allow user input during the running of the program. We use this input to make changes in the game.

A few things to note

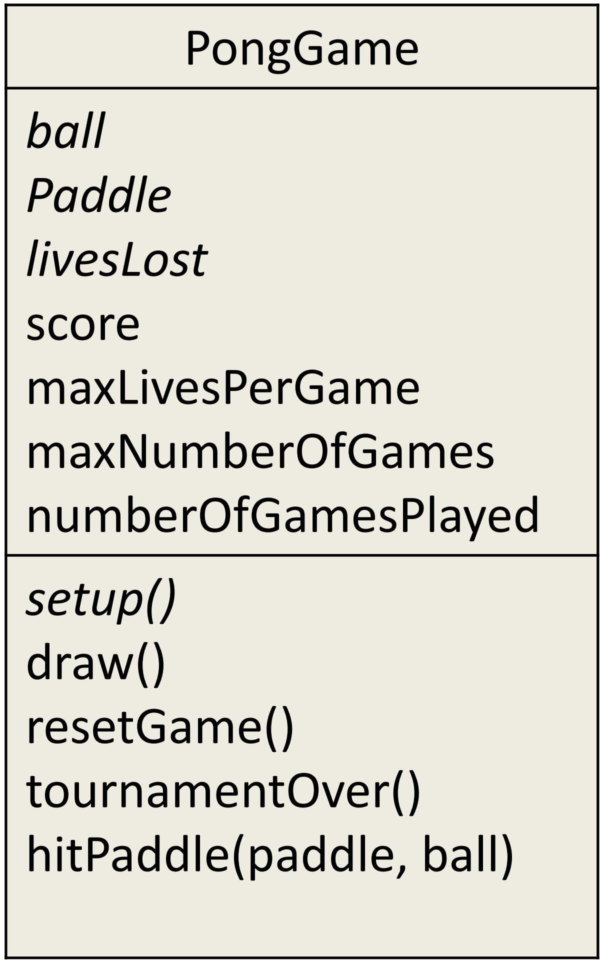
- We only use data input or data output in the Game(Driver) class.
- This is to ensure that the ‘user of classes’ (writer of the Driver) gets to decide how the data is input and output.
- This is why toString() is useful – it returns a string version of an object of a class – then the user can decide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

To use JOptionPane

```
import javax.swing.*;

//Objects required in the program
Ball ball;
Paddle paddle;
Player player;
:
```

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



Reading in maxNumberofGames

```
int maxNumberOfGames;
:
 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 used in the constructor directly.

maxNumberOfGames is read in

PongGame
ball *Paddle* *livesLost* score maxLivesPerGame maxNumberOfGames numberOfGamesPlayed

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

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 they have reached the max number of games as read in they will finish without being asked.

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();
  }
```

PongGame

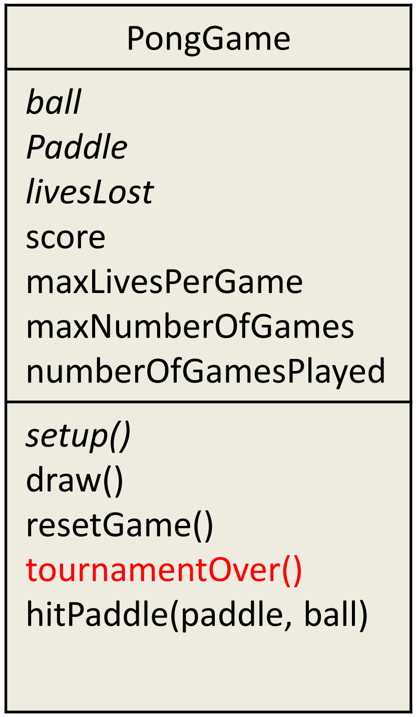
ball
Paddle
livesLost
score
maxLivesPerGame
maxNumberOfGames
numberOfGamesPlayed

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

We have added functionality here.

Improving look of output in 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();
  }
```



We have added functionality here.

Adding choice during the game

```
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();
 }
```

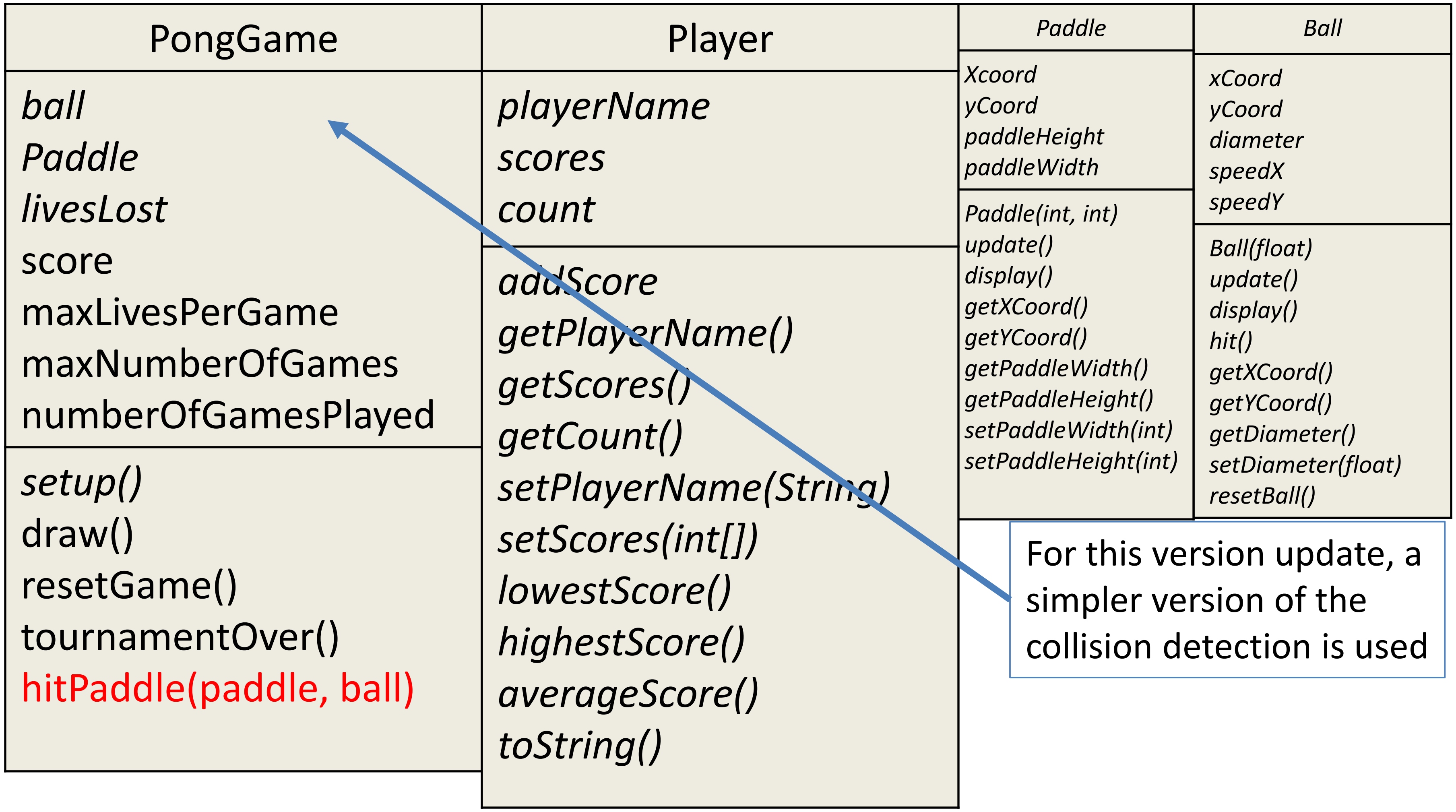
PongGame

ball
Paddle
livesLost
score
maxLivesPerGame
maxNumberOfGames
numberOfGamesPlayed

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

The same data is being output, just in a better way.

Classes in the PongGameV8.1



PongGame
ball *Paddle* *livesLost* score maxLivesPerGame maxNumberOfGames numberOfGamesPlayed

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

Player
playerName *scores* *count*

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

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()

For this version update, a simpler version of the collision detection is used

Changes to the collision detection algorithm

- In order to simplify the algorithm, some simplifications were made.
- This algorithm still works.
- It will not work if you move the paddle to the horizontal axis.
- This is often the case – the simpler algorithm is not as re-usable.
- The original algorithm would work for either axis.

Questions?

References

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