Game of Pong

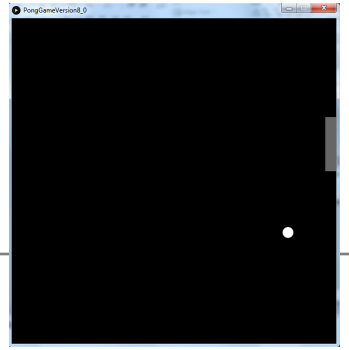
V7 Developing the game further

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

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

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

Pong Versions - introduction



v1 - **Ball moving** from left to right of screen. Can bounce off top or bottom

v2 - **Mouse controlling the Paddle**

---

v3 - **Collision detection** (ball bounces back). Changes made only to PongGame

v4 - **Game Over** (when 3 lives gone), Score (lives Lost). Output to Console. Changes made only to PongGame.

v5 - **Tournament** (no of games per tournament default is 5). Changes made only to PongGame.

---

v6 - new **Player class using arrays** (no statistics)

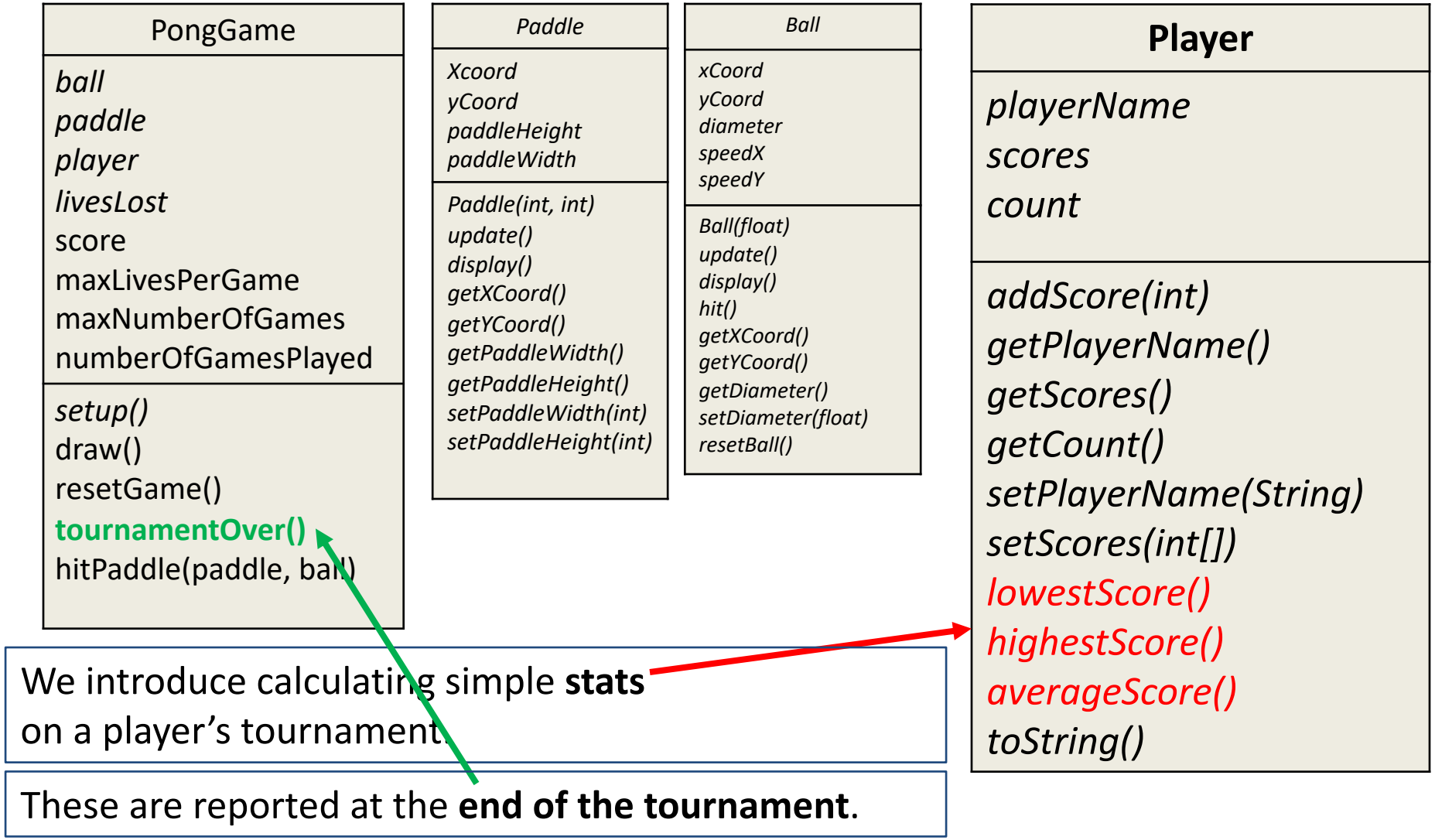
v7 - Player class using arrays (with **statistics** (Tournament Over - highest, lowest, average score))

v8 - **JOptionPane for I/O** instead of console

v9 - alternative algorithm using **Pythagoras Theorem**

Demo of
Pong Game V7.0

Classes in the PongGame**V7.0**



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 cl**ass
  - 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
playerName *scores* *count*
addScore *getPlayerName()* *getScores()* *getCount()* *setPlayerName(String)* *setScores(int[])* *lowestScore()* *highestScore()* *averageScore()* *toString()*

lowestScore()

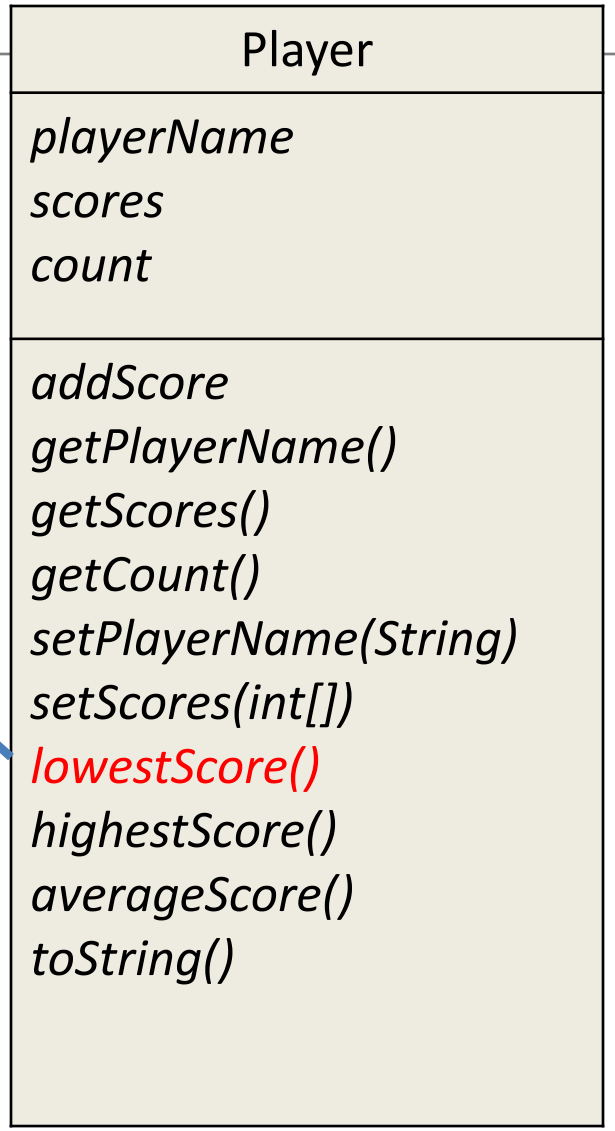
```
public int lowestScore()  {

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.



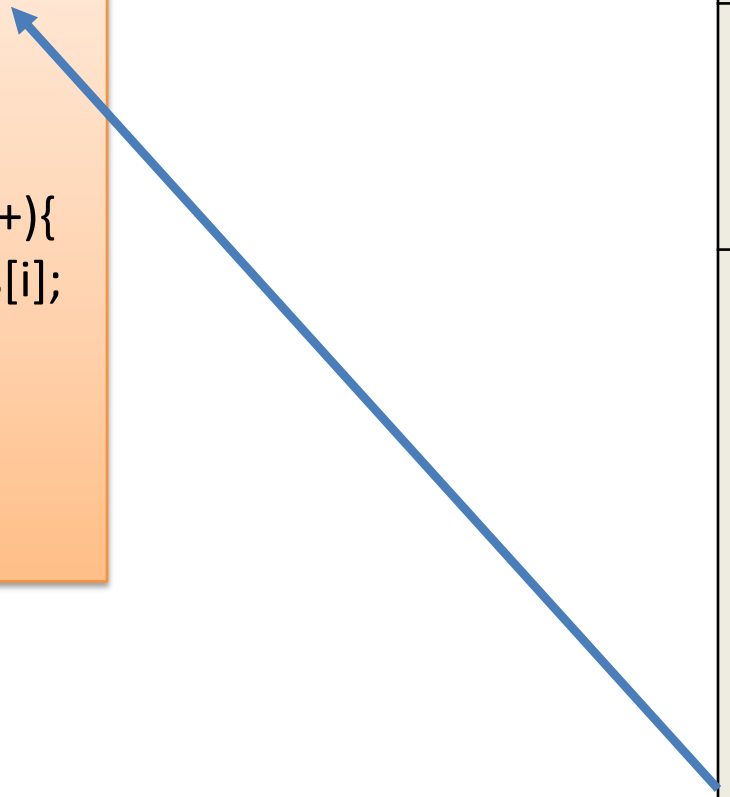
averageScore()

```
public int averageScore()  {

int total = 0;

for(int i = 0; i < count; i++){
     total = total +  scores[i];
 }

return total / count;
}
```



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

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();
}
```

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

This method calls the **stats methods** on the player object:
player.highestScore
player.lowestScore
player.averageScore

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.

Questions?

References

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