Some rights reserved

Processing
Pong 1

Coder Brixton
www.coderbrixton.com
Gareth Shapiro
www.garethshapiro.com
Maciej Falski

Let’s start.

Create a **new** sketch called **Pong**

and **save** it!

Add the two **functions** found in every Processing program.

setup

&

draw

pong/lesson1/step1

```
void setup() {
  size(800, 500);
}
void draw() {
  background(100);
}
```

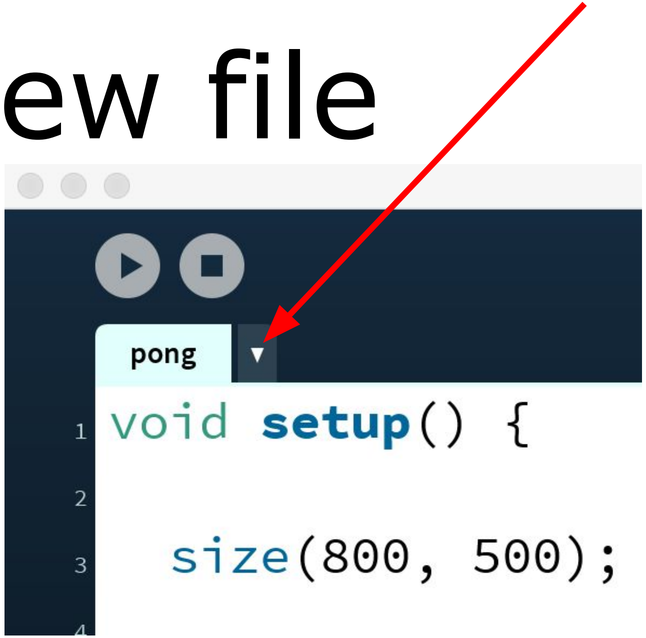
pong/lesson1/step1

The pong.pde file is the place where the code **starts** when you **run the program**.

It’s main job is to **create the game**.

The game logic is all contained in a **Game Class**.

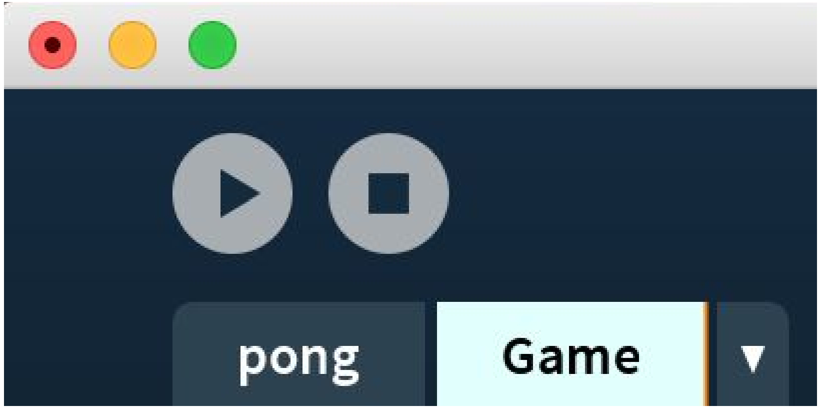
Create and save a new file called Game.



```
void setup() {

  size(800, 500);
```

pong/lesson1/step2

```
class Game {

  public Game() {



  }

}
```



This is called a
constructor

pong/lesson1/step2

Add two **functions** to the Game **class**

tick()

&

draw()

```
class Game {
    public Game() {
    }
     void tick() {
     }
     void draw() {
     }
}
```

functions inside a **class** are called **methods**.

pong/lesson1/step2

Now that the Game **class** has been **defined** we can create an **instance** of it.

We’ll do this in the *pong* file.

```
Game game;

void setup() {

  size(800, 500);

  game = new Game();

}
```

Define a **variable** called **game** of type **Game**

Create a **Game instance** and store it in **variable** called **game**

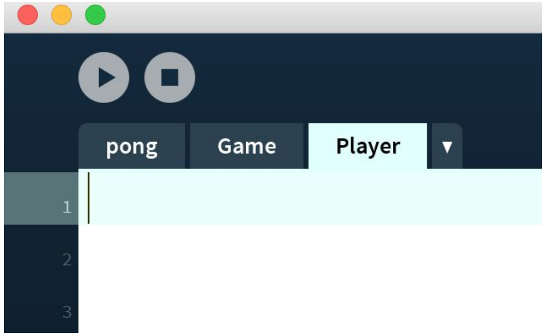
pong/lesson1/step2

```
Game game;

void setup() {

  size(800, 500);

  game = new Game();

}
```

Notice the link between the variable **type** and the name of the **class**.

Every game needs some players …

Let’s create a **Player class**.



pong/lesson1/step3

```
class Player {

  public Player() {



  }

}
```

Do you remember what this is called?

Add two **functions** to the Player **class**

tick()

&

draw()

```
class Player {
    public Player() {
    }
    void tick() {
    }
    void draw() {
    }
}
```

Do you remember what **functions** inside a **class** are called?

pong/lesson1/step3

Now that the Player **class** has been **defined** we can create two **instances** of it.

We’ll do this in the **Game class**.

```
class Game {

    Player player1;
    Player player2;

    public Game() {

        player1 = new Player();
        player2 = new Player();

    }

}
```

Define two **variables** of type **Player**

Create two **Player instances** and store them in the **variables**

pong/lesson1/step3

You have probably noticed that the Game **class** and Player **class** have both defined tick() and draw() **methods**.

The *pong* file also has a draw() **function** and remember this is keeps getting called in a loop.

Let’s connect them all together ….

Start with the **pong file**

pong/lesson1/step4

```
Game game;

void setup() {

    size(800, 500);

    game = new Game();
}

void draw() {

  background(100);

  game.tick();

}
```

pong/lesson1/step4

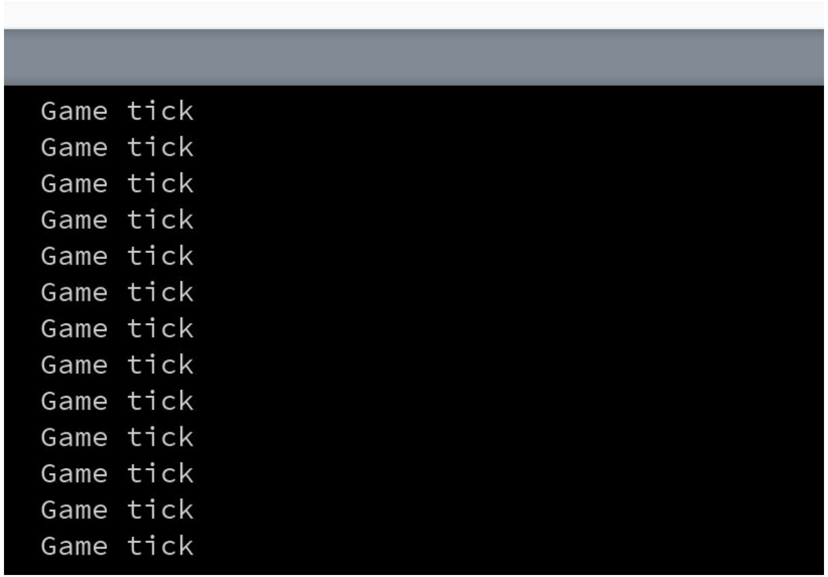
Now in the Game class let’s **add a test** to see if this has worked …

pong/lesson1/step4

```
public Game() {

    player1 = new Player();
    player2 = new Player();

}

void tick() {

    println("Game tick");
}
```

pong/lesson1/step4

If you run your program you should see …



```
Game tick
Game tick
Game tick
Game tick
Game tick
Game tick
Game tick
Game tick
Game tick
Game tick
Game tick
Game tick
Game tick
```

pong/lesson1/step4

Let’s connect the **Game** tick() **method** to the **Player** tick() **method** …

pong/lesson1/step4

```
public Game() {

    player1 = new Player();
    player2 = new Player();

}

void tick() {

    println("Game tick");

    player1.tick();
    player2.tick();

}
```

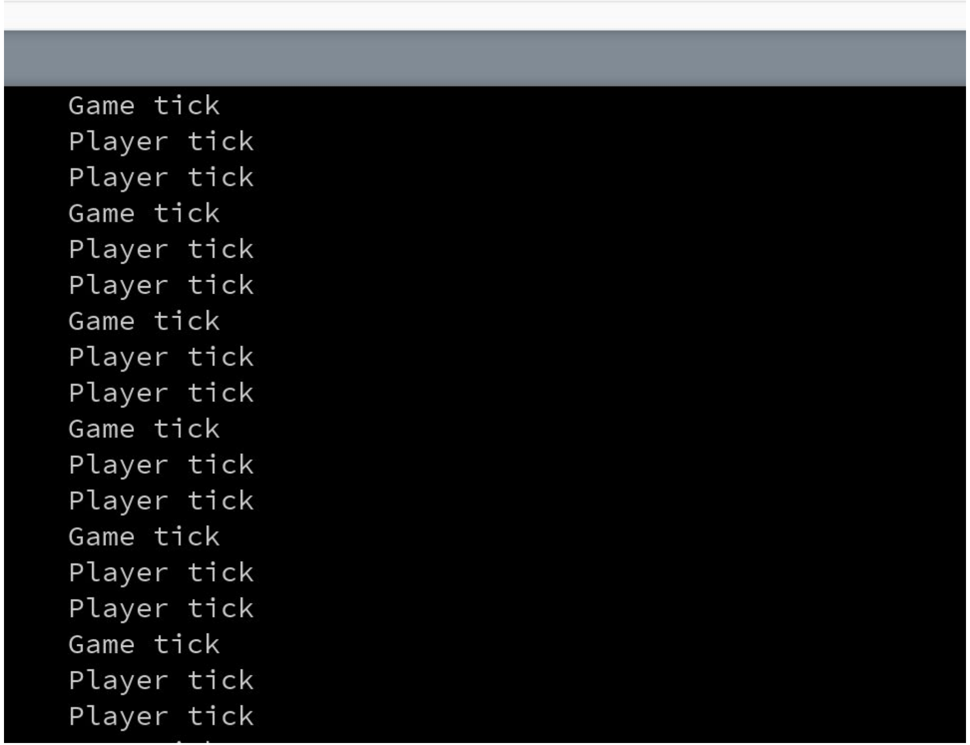
pong/lesson1/step4

Can you guess what the **test** is going to be?

```
class Player {

    public Player() {

    }

    void tick() {

        println("Player tick");
    }

    void draw() {

    }
}
```

pong/lesson1/step4

If you run your program you should see ...



```
Game tick
Player tick
Player tick
Game tick
Player tick
Player tick
Game tick
Player tick
Player tick
Game tick
Player tick
Player tick
Game tick
Player tick
Player tick
Game tick
Player tick
Player tick
```

Great.

We have created
a game loop.

pong/lesson1/step4

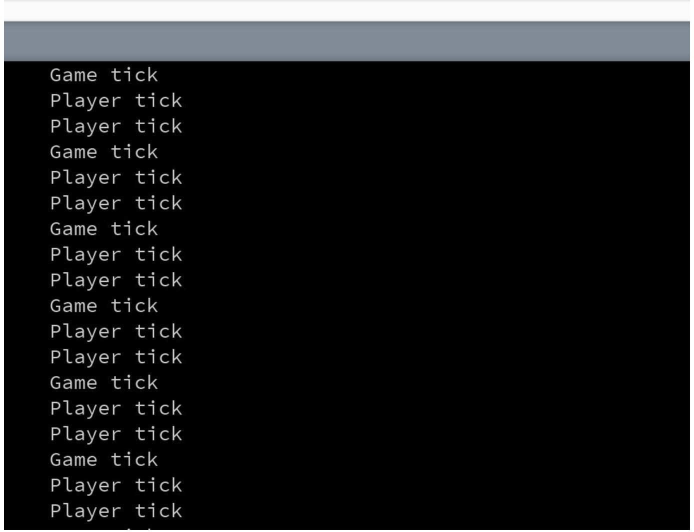
There is only one problem.

We can’t tell the **Player instances** apart.

Did you notice?

Both **instances** player1 and player2 both print Player tick

How can we make them behave **independently**?



```
Game tick
Player tick
Player tick
Game tick
Player tick
Player tick
Game tick
Player tick
Player tick
Game tick
Player tick
Player tick
Game tick
Player tick
Player tick
Game tick
Player tick
Player tick
```

Add a **property** called *side* of *type* **String** to the Player **class.**

```
class Player {

    String side;

    public Player(String side) {

        this.side = side;
    }

    void tick() {

        println("Player tick");
    }
    ...
}
```

variables inside a **class** are called **properties**.

pong/lesson1/step5

```
class Player {

    String side;

    public Player(String side) {

        this.side = side;
    }

     void tick() {

        println("Player tick");
     }
     ...
}
```

local scope

side **variable** does not live outside of the **initialiser**

pong/lesson1/step5

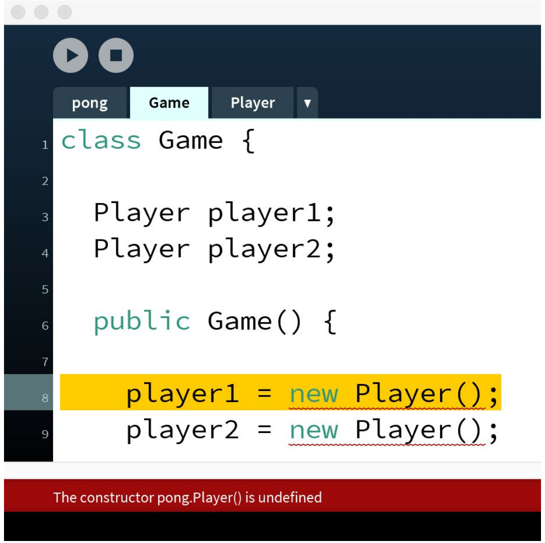
```
class Player {

    String side;

    public Player(String side) {

        this.side = side;
    }

     void tick() {

         println("Player tick");
     }
     ...
}
```

class scope

side **property** lives everywhere in the Player **class**

pong/lesson1/step5

What happens when you run the program?



```
class Game {

  Player player1;
  Player player2;

  public Game() {

    player1 = new Player();
    player2 = new Player();
```

The constructor pong.Player() is undefined

pong/lesson1/step5

The **constructor** pong.Player is undefined.

What’s the difference between the *Player* class’ **constructor** and how the *Player* class is **intialised** in the *Game* class?

pong/lesson1/step5

We added one **argument** to the **constructor** …

```
public Player(String side) {

   this.side = side;

}
```

pong/lesson1/step5

We need to supply a **value** for the **argument** ..

```
public Game() {

    player1 = new Player("left");
    player2 = new Player("right");

}
```

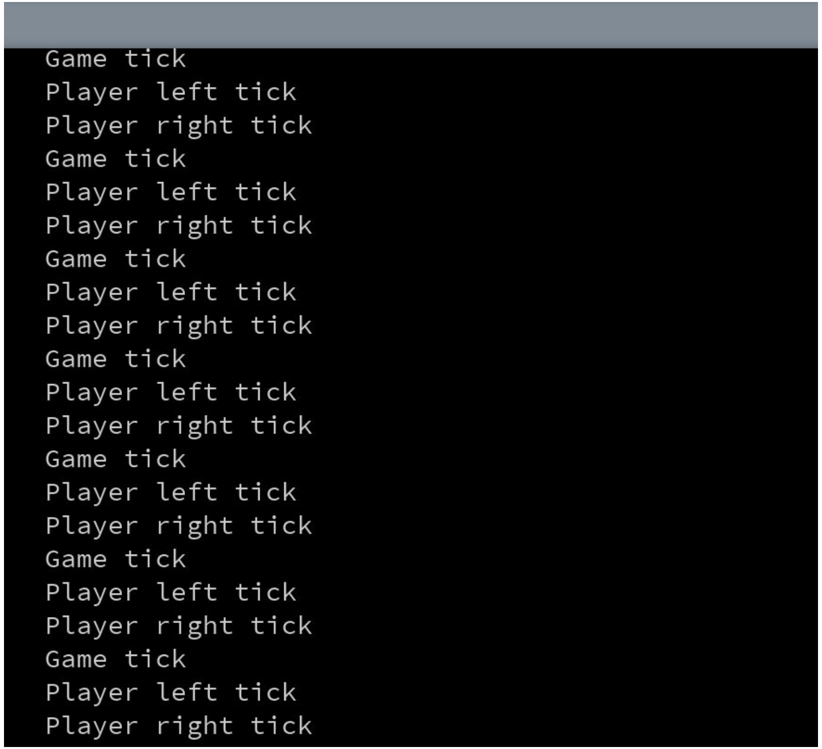
pong/lesson1/step5

Now that we have **passed a value** for to the Player object when we **intialised** it we should do something with it.

```
class Player {

    String side;

    public Player(String side) {

        this.side = side;
    }

    void tick() {

        println("Player " + this.side + " tick");
    }
    ...
}
```

pong/lesson1/step5

Run the program and let’s test the code …



```
Game tick
Player left tick
Player right tick
Game tick
Player left tick
Player right tick
Game tick
Player left tick
Player right tick
Game tick
Player left tick
Player right tick
Game tick
Player left tick
Player right tick
Game tick
Player left tick
Player right tick
Game tick
Player left tick
Player right tick
```

pong/lesson1/step5

Now that we know which Player is on the left and which is on the right let’s draw them.

pong/lesson1/step5

```
class Player {

    String side;

    int barLength = 140;
    int barWidth = 20;

    int x;
    int y = height/2 - barLength/2 ;
     …
}
```

pong/lesson1/step5

```
public Player(String side) {

    this.side = side;

    if (side == "left") {

        x = 30;

    } else {

        x = width - 30;
    }

}
```

pong/lesson1/step5

```
class Player {
    …

    void draw() {

        if (this.side == "left") {

            rect(x - barWidth, y, barWidth, barLength);

        } else {

            rect(x, y, barWidth, barLength);
        }
    }
  }
```

pong/lesson1/step5

Run the code,
does it work?

NO!

Why not?

pong/lesson1/step5

We have added drawing code to the draw() **method** of the Game **class** but we are not **calling** the method.

Where should we **call**

Player.draw()

?

pong/lesson1/step5

```
public Game() {

    player1 = new Player("left");
    player2 = new Player("right");

}

void draw() {


    player1.draw();
    player2.draw();

}
```

pong/lesson1/step5

Run the code,
does it work?

pong/lesson1/step5

NO!

Why not?

Where is Game.draw() being **called** from?

Where should we **call**

Game.draw()

?

pong/lesson1/step5

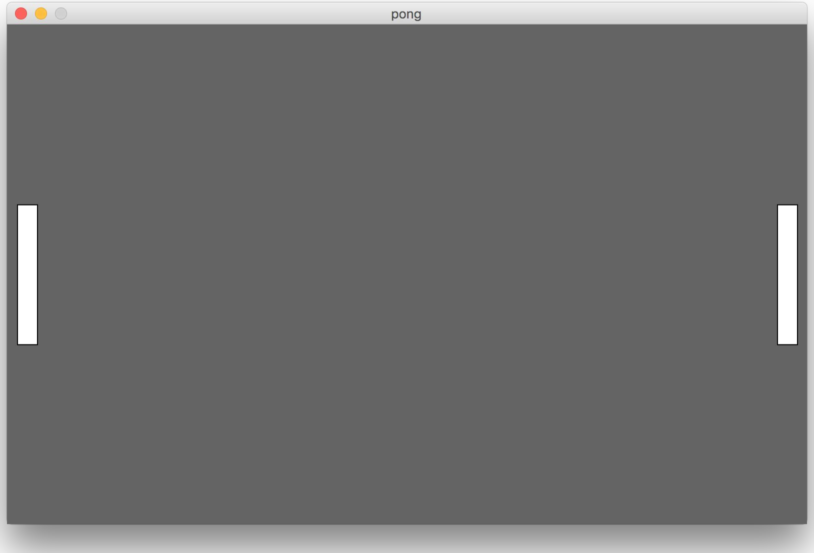
In the pong file!

```
void setup() {

    size(800, 500);

    game = new Game();
}

void draw() {

  background(100);

  game.tick();
  game.draw();

}
```

pong/lesson1/step5

Run the code,
does it work?

Woo Hoo!



pong/lesson1/step5

Let’s make one small visual adjustment and remove the black **stroke** outline from the Players.

In the pong file!

```
void setup() {

    size(800, 500);

    noStroke();

    game = new Game();
}
```

pong/lesson1/step5

So the players are drawing correctly but we could make an improvement.

We are using a **String value** to tell them part.

```
public Player(String side) {

   this.side = side;

}
```

pong/lesson1/step6

Strings are words

It’s easy to make a typing mistake with words.

```
public Game() {
    player1 = new Player("left");
}
class Player {
    void draw() {
        if (this.side == "lefty") {
            rect(x - barWidth, y, barWidth, barLength);
        } else {
            rect(x, y, barWidth, barLength);
        }
    }
}
```

These don’t match!

This is a **BUG!**

What happens if this code runs?

pong/lesson1/step6

Let’s make this better by using a **type** to make our **Player instances** independant.

pong/lesson1/step6

```
enum Side {
  LEFT, RIGHT
}
class Player {
        Side side;
       public Player(Side side) {
              this.side = side;
              if (side == Side.LEFT) {
                     ...
              }
       }
}
```

pong/lesson1/step6

```
class Player {
    void draw() {
        if (this.side == Side.LEFT) {
            rect(x - barWidth, y, barWidth, barLength);
        } else {
            rect(x, y, barWidth, barLength);
        }
    }
}
```

pong/lesson1/step6

What happens when you run the program?

The constructor pong.Player(String) is undefined

What’s the difference between the *Player* class’ **constructor** and how the *Player* class is **intialised** in the *Game* class?

```
public Game() {

    player1 = new Player("left");

}
```

pong/lesson1/step6

The definition expects a **Side type** and we are supplying a **String type**.

The types have to match.

```
public Game() {

    player1 = new Player(Side.LEFT);
    player2 = new Player(Side.RIGHT);

}

void tick() {

    println("Game tick");

    player1.tick();
    player2.tick();

}
```

pong/lesson1/step6

Run the code,
does it work?

GAME ON!

Now we can only supply

SIDE.LEFT

or

SIDE.RIGHT

to the Player **constructor**

Time to get our player instances moving

pong/lesson1/step7

First let’s clean up the println() calls inside Game and Player

pong/lesson1/step7

```
class Game {
    ...
    void tick() {

        ~~println("Game tick");~~
    }
    ...
}
class Player {
    ...
     void tick() {

        ~~ println("Player " + this.side + " tick");~~
     }
     ...
}
```

pong/lesson1/step7

Let’s add **keyboard event handlers** to the pong file

pong/lesson1/step7

```
void keyPressed() {
    if (keyCode == UP) {
        println("moveUpPlayer2()");
    } else if (keyCode == DOWN) {
        println("moveDownPlayer2()");
     } else if (key == 'w') {
        println("moveUpPlayer1()");
     } else if (key == 's') {
        println("moveDownPlayer1()");
    }
}
```

pong/lesson1/step7

```
void keyReleased() {
    if (keyCode == UP || keyCode == DOWN) {
        println("stopPlayer2()");
    } if (key == 'w' || key == 's') {
        println("stopPlayer1()");
    }
}
```

pong/lesson1/step7

Test your code.

Do you see what you would expect in the console?

We need to do a bit more than print to the console.

How do we get to our **Player instances** from the pong file?

pong/lesson1/step8

We use the **Game instance**!

pong/lesson1/step8

```
class Game {

    void moveUpPlayer1() {

    }

    void moveDownPlayer1() {

    }

    void stopPlayer1() {

    }
}
```

pong/lesson1/step8

```
class Game {

    void moveUpPlayer2() {

    }

    void moveDownPlayer2() {

    }

    void stopPlayer2() {

    }
}
```

pong/lesson1/step8

The **Game instance** has reference to the **Player instance** but first we need to create some **methods** on Player so that other classes can move them.

pong/lesson1/step9

```
  class Player {

      public void moveUp() {

      }

      public void moveDown() {

      }

      public void stop() {

      }

}
```

pong/lesson1/step9

Let’s connect the **keyboard event handlers** in the pong file to the relevant **Game class methods** (luckily we named them reasonably) ..

pong/lesson1/step10

```
void keyPressed() {
    if (keyCode == UP) {
        game.moveUpPlayer2();
    } else if (keyCode == DOWN) {
        game.moveDownPlayer2();
     } else if (key == 'w') {
        game.moveUpPlayer1();
     } else if (key == 's') {
        game.moveDownPlayer1();
    }
}
```

pong/lesson1/step10

This means **or**

```
void keyReleased() {
    if (keyCode == UP || keyCode == DOWN) {
        game.stopPlayer2();
    } else if (key == 'w' || key == 's') {
        game.stopPlayer1();
    }
}
```

pong/lesson1/step10

Run the code,
does it work?

NO!

Why not?

pong/lesson1/step10

The **keyboard event handlers** call **Game class methods** and what do they call …?

pong/lesson1/step10

NOTHING!

What should the **Game methods** that are **called** by the **keyboard event handlers** do?

The should **call methods** on the **Player instances** …

Let’s get to it.

```
 class Game {

    void moveUpPlayer1() {
        player1.moveUp();
    }

    void moveDownPlayer1() {
        player1.moveDown();
    }

    void stopPlayer1() {
        player1.stop();
    }
}
```

pong/lesson1/step10

```
 class Game {

   void moveUpPlayer2() {
      player2.moveUp();
   }

   void moveDownPlayer2() {
      player2.moveDown();
   }

   void stopPlayer2() {
      player2.stop();
   }
}
```

pong/lesson1/step10

Run the code,
Do the players move?

NO!

Why not?

Take a look in the **Player class** …

moveUp()
moveDown()
moveStop()

All these **methods** are empty
We haven’t written any move code….

In a later session we are going to add some acceleration to our Players’ movement.

Not today but we can make a start.

Acceleration works like this :

The longer you press a key the faster a Player will move in that direction.

For this we are going to have to know if the the key is still down or has it gone up yet.

This type of information is called **state**.

Quite often when **state is recorded** we used something called a **flag**

Let’s add an *isMoving* **flag**

```
 class Player {

     boolean isMoving = false;

    public void moveUp() {
        this.isMoving = true;
    }

    public void moveDown() {
        this.isMoving = true;
    }

    public void stop() {
        this.isMoving = false;
    }
}
```

Setting a **default state** now saves having to test whether one has been set later ..

pong/lesson1/step11

We have a similar detail to the **position** of a Player when we think about the **direction** a Player can move in.

The Player can have two positions :

LEFT
RIGHT

The Player can move in two directions :

UP
DOWN

How can we write code that handles this detail in a similar way?

We can use another **enum**!

enum is short for enumeration….

pong/lesson1/step11

```
enum MovementDirection {
    UP, DOWN
}

class Player {

    boolean isMoving = false;
    MovementDirection direction = MovementDirection.UP;
}
```

Setting a **default state** now saves having to test whether one has been set later ..

pong/lesson1/step11

```
  class Player {

    ...
    public void moveUp() {

        this.isMoving = true;
        direction = MovementDirection.UP;
    }

    public void moveDown() {
        this.isMoving = true;
        direction = MovementDirection.DOWN;
    }

    ...
}
```

pong/lesson1/step11

Can you see the **connection** between **keyboard events** and **changes to state** in the game?

Now that we have built the **basic architecture** and **Player logic** there is only one thing left to do...

pong/lesson1/step12

```
class Player {
    ...
    void tick() {

        if (isMoving) {

            if (direction == MovementDirection.UP) {

                y -= 5;

            } else {

                y += 5;
            }
        }
    }
}
```

pong/lesson1/step12

At last … !

Two moving Players

(If you have no bugs)!

pong/lesson1/step12

If you are on a Mac you may notice sometimes the Players “stick” a bit when you move …

If you are on a Mac open a terminal and type in this command :

```
defaults write -g ApplePressAndHoldEnabled
-bool false
```

And restart Processing.

This will improve the keyboard interaction with Processing …

See you next time …