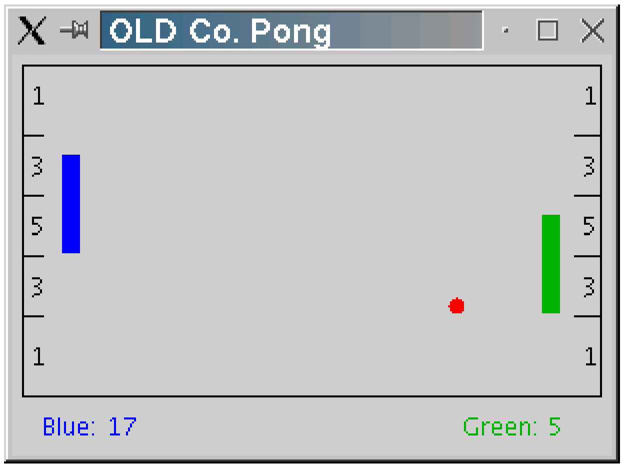
Object-Oriented Language Development Company

The OLD Co Pong Game: Executive Summary

The Object-Oriented Language Development Company (OLD Co.) has begun development of a radically new computer game, with the wholly original name of Pong.



Pong is a two player game in which each player attempts to prevent the pong-ball from reaching the wall behind his pong-paddle. A player deflects the pong-ball before it reaches the wall by placing his pong-paddle into path of the pong-ball. This causes the ball to bounce off of the paddle in the opposite direction. If the ball reaches the wall behind a player's paddle, then the opponent scores. The number of points scored is dependent on the region of the wall contacted by the ball. The closer to the center of the wall the greater the number of points scored.

The high-paid software engineers at OLD Co. have spent years designing the Pong program. They have identified the classes that describe the objects the program will use. They have carefully defined the public interfaces of each of the objects. The specifications for these classes were handed off to the programmers who completed the majority of the implementation. However, just before the project was completed, all of the programmers were offered huge salaries by Google and they simply quit on the spot!

I've agreed to write these classes for OLD Co. But being, ever the educator, I decided that writing these classes would be a good experience for you. Of course, I get to keep the check from OLD Co!) The remainder of this document describes the classes you'll need to write to complete the Pong game.

The PongBall Class:

Description:

```
PongBall
-------------------------------------------
x       xVelocity
y       yVelocity
-------------------------------------------
PongBall(int initX, int initY
         int initXVel, int initYVel)
-------------------------------------------
void bounceX()
void bounceY()
int getX()
int getY()
void move()
```

The `PongBall` class describes the object which the Pong application uses to represent the ball that bounces around the screen. The following class diagram illustrates the information and operations that are defined by the `PongBall` class.

When the Pong application is executed it creates a new `PongBall` object that is positioned at the center of the playing field. Approximately every 10th of a second the Pong application calls the `move()` method on its `PongBall` object causing the ball to update its position. Following the call to `move()` the Pong application uses the `getX()` and `getY()` methods to find the new location of the ball. If the Pong application determines that the ball would have collided with a wall or a paddle it will invoke the `bounceX()` or the `bounceY()` method to change the horizontal or vertical direction of the ball. Finally, the Pong application will draw the ball on the screen at its new location and the process will be repeated again in another 10th of a second.

Implementation:

To implement the `PongBall` class you will need to define instance data for the x and y position of the ball as well as the velocity of the ball in the x and y directions. Your implementation of the `PongBall` class must meet these specifications, so please study them carefully.

`Constructor`: construct a new `PongBall` with the specified initial position and velocity.

Construction Parameters:

- **`initX`** - the initial X position of the new `PongBall`.
- **`initY`** - the initial Y position of the new `PongBall`.
- **`initXVel`** - the initial velocity of the new `PongBall` in the X direction.
- **`initYVel`** - the initial velocity of the new `PongBall` in the Y Direction.

Public Interface Specification:

- **`void bounceX()`** - Reverse the X direction of this `PongBall`. This method is invoked by the Pong application each time this `PongBall` collides with a vertical obstruction such as a wall or paddle. The X direction can be reversed by changing the sign of the X velocity.

- **`void bounceY()`** - Reverse the Y direction of this `PongBall`. This method is invoked by the Pong application each time this `PongBall` collides with a horizontal obstruction such as a wall or the top/bottom edge of paddle. The Y direction can be reversed by changing the sign of the Y velocity.

- **`int getX()`** - Return the current X position of this `PongBall`.

- **`int getY()`** - Return the current Y position of this `PongBall`.

- **`void move()`** - Move this `PongBall` according to its current velocity. This method is invoked at a regular rate by the Pong application in order to update the position of the ball. The X position changes by the velocity in the X direction and the y position changes by the velocity in the Y direction.

Integration:

You don't need to do anything special to integrate your new code into the `PongGame` project. Simply run the `Pong` program and your new class will replace the old, useless `PongBall` class provided by the original programmers. When the program starts, press the "b" key and the ball should start bouncing. (Pressing "b" again will speed it up.)

The PongScore Class:

Description:

The `PongScore` class describes the object that the Pong application uses to keep track of the score for each player. The following class diagram illustrates the information and operations that are defined by the `PongScore` class.

PongScore
score
PongScore()
int getScore() void scorePoints(int points)

When the Pong application is executed it creates two `PongScore` objects, one for the blue player and one for the green player. Each time the ball contacts the end wall behind the blue paddle, the `scorePoints(...)` method of the green player's `PongScore` object is invoked, and vice versa. When invoking the `scorePoints(...)` method, the Pong application passes it an argument (or parameter) corresponding to the number of points indicated in the region of the wall that was contacted. The Pong application will then invoke the `getScore()` method to determine the score to be displayed beside the player's name below the playing field.

Implementation:

Implement the `PongScore` class in your DrJava project. Your implementation must meet the specifications given below.

`Constructor`: construct a new `PongScore` with an initial score of zero points. There are no construction parameters.

Public Interface Specification:

- **`int getScore()`** - Return the current score of this `PongScore` object.

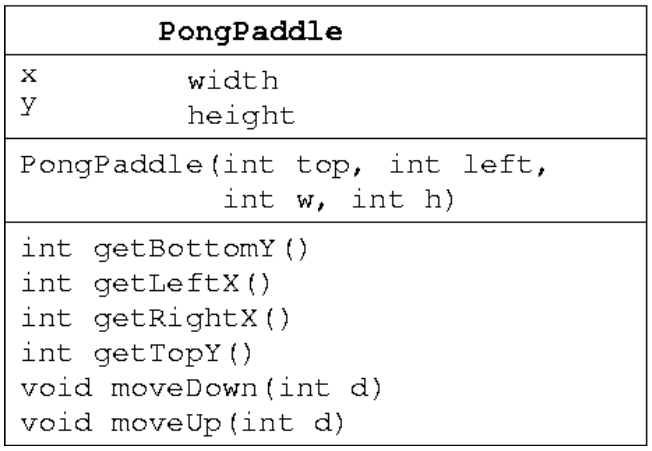
- **`void scorePoints(int points)`** - Increase the current score of this `PongScore` object by points.
    - **`points`** - the number of points to add to the score.

Integration:

When you run the Pong application in DrJava and press the "B" key, the ball will again begin to bounce around the playing field. However, now each time the ball contacts the end wall the opponent's score will be incremented by the appropriate number of points.

The PongPaddle Class:

Description:



```
PongPaddle
-----------------------------------
x            width
y            height
-----------------------------------
PongPaddle(int top, int left,
              int w, int h)
-----------------------------------
int getBottomY()
int getLeftX()
int getRightX()
int getTopY()
void moveDown(int d)
void moveUp(int d)
```

The `PongPaddle` class describes the objects that the Pong application uses to keep track of the size and location of the pong paddles. The following class diagram illustrates the information and operations that are defined by the `PongPaddle` class. Each `PongPaddle` has an `(x,y)` position that indicates the position of the top-left corner of the paddle. Each `PongPaddle` also has a `width` and a `height`.

When the Pong application is run it creates two new `PongPaddle` objects: one for the blue player positioned at the left end of the playing field, the other for the green player positioned at the right end of the playing field. When the blue player presses the "A" key, the Pong application invokes the `moveUp(...)` method of the blue player's `PongPaddle` object. Conversely, when the blue player presses the "Z" key the Pong application invokes the `moveDown(...)` method. Similar actions are taken using the green player's `PongPaddle` object when the "M" and "K" keys are pressed. The Pong application then uses the `getLeftX`(), `getTopY()`, `getRightX()` and `getBottomY()` methods to determine where on the screen the blue and green paddles should be drawn. These methods are also used by the Pong application to determine when the ball collides with one of the paddles.

Implementation:

Implement the `PongPaddle` class in your DrJava project. Your implementation must meet the specifications given below.

`Constructor`: Construct a new `PongPaddle` at the specified position with the specified width and height.

Construction Parameters:

- **`top`** - `the Y coordinate of the top left corner of the new PongPaddle.`
- **`left`** - `the X coordinate of the top left corner of the new PongPaddle.`
- **`w`** – `the width of the new PongPaddle, measured in pixels.`
- **`h`** – `the height of the new PongPaddle, measured in pixels.`

Public Interface Specification:

- **`int getBottomY()`** - Return the Y coordinate of the bottom edge of this `PongPaddle`.

- **`int getLeftX()`** - Return the X coordinate of the left edge of this `PongPaddle`.

- **`int getRightX()`** - Return the X coordinate of the right edge of this `PongPaddle`.

- **`int getTopY()`** - Return the Y coordinate of the top edge of this `PongPaddle`.

- **`void moveDown(int pixels)`** - Move the position of this `PongPaddle` down by the specified number of pixels..
  - **`pixels`** - the number of pixels to move down.

- **`void moveUp(int pixels)`** - Move the position of this `PongPaddle` up by the specified number of pixels..
  - **`pixels`** - the number of pixels to move up.

Integration:

Once you have completed your `PongPaddle` class, run the Pong application in DrJava. Press the "B" key to start the ball bouncing as before. Use the "A" or "Z" keys to control the blue paddle. Similarly, the "K" or "M" keys control the green paddle.

Congratulations! You now have a complete working Pong application! Next week, we'll look at **testing** it to see if it's correct!.