Text on Block

We know how to paint text on a canvas
We want the number of lives to show in the center of our blocks
Much simpler than a full-blown text component

Here is what our block will do

- A block is created with a specified number of lives
- A text painter is set to paint that number
- On render() that number is painted inside the block rectangle
- Whenever the lives variable changes, we need to set up new text to paint

Here is our new instance variable for painting the lives

```
TextPainter tp = TextPainter(
  textAlign: TextAlign.center,
  textDirection: TextDirection.ltr,
  textScaleFactor: 1.5,
);
```

We pick a scale of 1.5 because that gives a good sized number

Tracking Lives

Remember that a text painter needs to compile the text so it can be painted to a canvas

This has to happen whenever to lives variable changes

How can we make sure this happens?

Could make a method changeLives() that does this

But someone could still change the lives variable

We do this with getters and setters

We fool the user of our object into thinking there is a lives variable

They set something like block.lives = 10

It actually calls a setter method – doesn't just change a variable

Hidden Lives

First, we need to hide the instance variable lives so that no one can set it directly

```
int _lives;  // how many hits until the block dies
```

By putting an underscore before a variable, we make it hidden so no one outside the class code can change it.

You cannot do block._lives = 10; it will fail.

Next, we make a lives setter that makes it appear that we have an instance variable called lives

```
void set lives (int l) {
}
```

Now, if someone uses the .lives property of the object, it actually calls the lives() setter which can do anything we want.

Finally, you put the code inside the setter that you want it to do

It sets the _lives variable and sets up the text painter with the new lives value to paint

```
void set lives (int l) {
  _lives = l;
  TextSpan span = TextSpan(text: _lives.toString(), style: TextStyle(color:
Colors.black));
  tp.text = span;  // text to draw
  tp.layout(minWidth: position.width, );
}
```

Outside the object, you change the lives by using the lives setter

block.lives = 10;

Inside the class code itself, you call the setter using this.lives = 123;

Reading Lives

We have a way of setting _lives with the lives setter

How do we read the value of _lives?

We cannot do block._lives since _lives is private;

We make a getter for lives

Normally, a getter just returns a simple variable, so it has a shorthand.

```
int get lives => _lives;
```

If the method we want to call only has one line, we can use => followed by what we want to return. If you wanted to do something more complicated, you could use a full method

```
int get lives {
  // do some stuff
  return _lives;
}
```

Back to Painting Lives

In our lives setter, we set up the text painter.

Remember, we need a text span and a style to describe the text

A simple style sets the color to black

We pass _lives.toString() as the text to paint

You have to pass a string, not an integer

Every object has a toString() method to turn it into a string

We compile the text and it is ready to paint in our render() method

Rendering the Block

We have 2 things to do in our render method

- Fill in the block rectangle with the block color
- Paint the lives number on top of the colored block

```
void render(Canvas c) {
  // draw the block and then the text on top
  c.drawRect(position, paint);
  tp.paint(c, position.translate(0, 10).topLeft);
}
```

Our drawRect() has been seen before.

We paint our lives text. If we painted it in the block rectangle, the number would be right at the top edge. We use the translate() method on the rectangle to make a new rectangle that is 10 pixels down from the original one. X=0 so it doesn't move left or right.

The topLeft() method on a rectangle gives us the top left point of the block, shifted down by 10.

Try it Out

Use the same block you added when you first made the Block component.

You will set a block with a 10 in the middle.

Change the lives in the game before you add it. The new number will now show.

The code can be found here:

https://github.com/shawnlg/flutter_ball/tree/13_block

Next time, we will make a block that can be dragged around the screen.

In our game, that will be our aiming block to control how the ball bounces.