Title Page

Project	Portrait Tree
Chapter	01
Content	The Start of the production code in a single file of PHP.

Chapter Overview

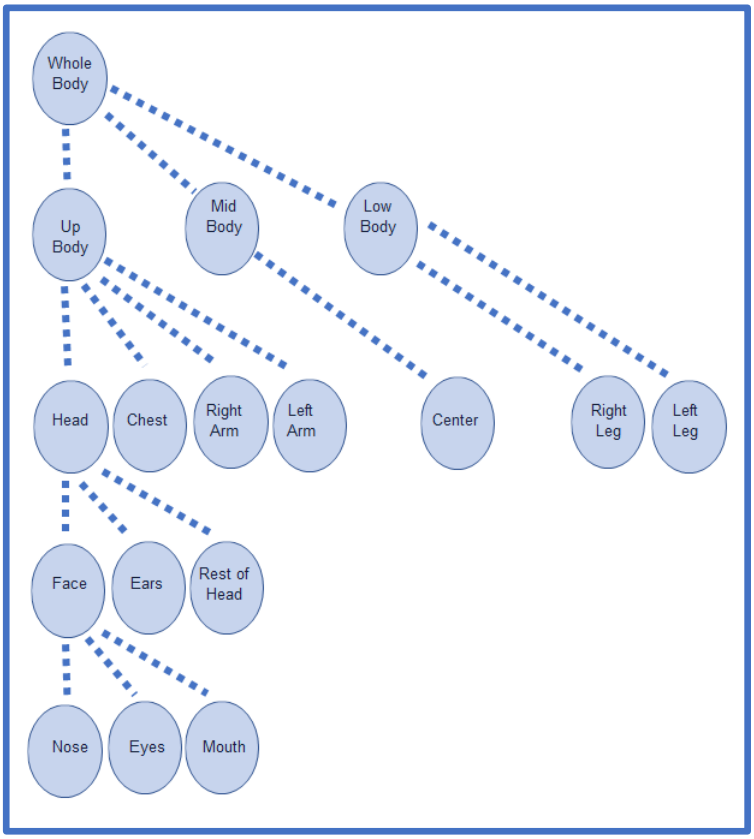
The "Objective" presents the current goal within its widest context. When the first code-sample runs, its output contributes to the current goal. Explaining the output, makes the goal easy to follow.

The code produced for this stage, prototypes a single Public Method. The output from here, shows how the method fits the goal and what more is needed to support it.

Objective

The project itself MUST have some sort of apex, which is replaced by progressively deeper levels until the last one is reached. Before mentioning any code, it helps to look at this:

Bubble Chart



To introduce the terminology, “UpBody, Midbody, LowBody” is only one row below the **apex** and the final two levels you can see are “Face, Ears, RestofHead”, followed by “Nose, Eyes, Mouth”

Thinking in a spreadsheet way, it is convenient to think of items in the same level being named A, B, C… Even on the same row, logical gaps show which items belong together. The following represents the intended output and what it means.

Meaning of output

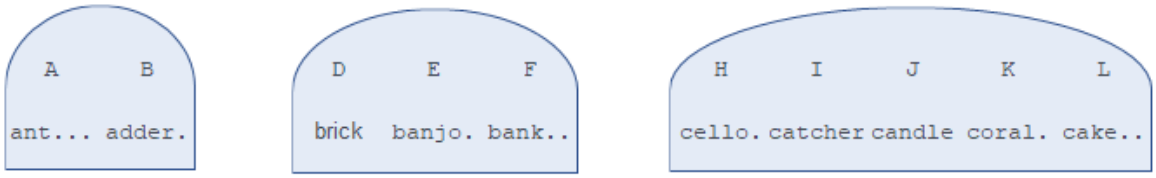
This demo for this stage comes from a new example. It is hard-wired, to give words starting with a, b, or c. The top left hand corner shows what the console program produces and the rest of the picture shows what that means.

```
==> < A >ant..
==> < B >adder
==> < D >brick.
==> < E >banjo.
==> < F >bank..
==> < H >cello.
==> < I >catcher
==> < J >candle
==> < K >coral.
==> < L >cake..
```

	A	B		D	E	F		H	I	J	K	L	
	ant...	adder.		brick	banjo.	bank..		cello.	catcher	candle	coral.	cake..	

Wishful Thinking

With some imagination, you could even see the “meaning of output” as:



Current Goal

In summary, members of the **current level** form the destination row taking account of gaps as explained above. Following this, each one of these members is replaced by its “children”, so that they in-turn become the new **current level**.

There is more work to do, but at this stage the focus is on pushing out each row as we prepare for the next one.

Highlights of Actual code

The part of this work that stays within future development is the function, **nextLevel.** Here, it is shown as a public method from the class **LinoBlock**.

Main Program running LinoBlock

```
145  } // End Class
146
147  $feed = array('Still',' Not','Using','it');
148  $objX = new LinoBlock($feed);
149  $objX->spyOn_Bench();
150  $delivaryPair = $objX->nextLevel();
151
```

The main Program represents the property workbench and then represents the output of nextLevel itself.

The Public Method

```
100
101 ▼          public function nextLevel(){
102                $EOK = $this->EndOfKey;
103
104                $pushedLevel =$this->indexedLevel();
105                $delivaryPair = [];
106                $pos = 64 ; # One before capital ** A **
107                $addrContent = [];
108
109 ▼              foreach ($pushedLevel as $key => $content) {
110                    $pos = $pos +1;
111 ▼                  if ($content<>$EOK) {
112                            $addr= strval(chr($pos));
113                            $addrContent = array($addr,$content);
114                            array_push($delivaryPair, $addrContent);
115                    } // End if
116                }// End for
117
118                return $delivaryPair;
119              } // End Func
120
```

The method calls **indexedLevel** which essentially brings in the right-hand column of data from the internal property, **workbench**.

Looking at the output

Questions about the Demo

The working program demonstrates a single level of output from the workbench. From here, we can consider answers for several questions related to the work until now:

- How do we get to the next level?
- How do we know when to stop?

Of course, the current object which comes from LinoBlock must start by answering:

- How do we start at the apex and begin the first lines?

Answering Questions

The data-environment implied by the hard-wired code has two levels. There are more than three levels required to support the picture and the two logical datasets are shown here.

Letters

a	ant
a	adder
b	brick
b	banjo
b	bank
c	cello
c	catcher
c	candle
c	coral
c	cake
ant	antler
ant	ant hill
adder	address
adder	adding
adder	addition

Whole Body

Whole	Up Body
Whole	Mid Body
Whole	Low Body
Up Body	Head
Up Body	Chest
Up Body	Right Arm
Up Body	Left Arm
Mid Body	Center
Low Body	Right Leg
Low Body	Left Leg
Head	Face
Head	Ears
Head	Rest of Head
Face	Nose
Face	Eyes
Face	Mouth

To proceed a single level further, focus moves forward one generation. Following the letters table: “ant” has children and “adder” has children. By experimenting with the code, we get clues for answering the remainder.

Procedure affects the workbench

Existing Code

col-0	col-1	col-2	col-3	col-4	col-5	
0	0	a	0	ant	}	==> < A >ant
0	0	a	0	adder	}	==> < B >adder
0	0	b	0	brick	}	==> < D >brick
0	0	b	0	banjo	}	==> < E >banjo
0	0	b	0	bank	}	==> < F >bank
0	0	c	0	cello	}	==> < H >cello
0	0	c	0	catcher	}	==> < I >catcher
0	0	c	0	candle	}	==> < J >candle
0	0	c	0	coral	}	==> < K >coral
0	0	c	0	cake	}	==> < L >cake

One More Gen.

col-0	col-1	col-2	col-3	col-4	col-5	
0	0	ant	0	antler	{	==> < A >antler
0	0	ant	0	antivirus	{	==> < B >antivirus
0	0	ant	0	ant hill	{	==> < C >ant hill
0	0	adder	0	address	{	==> < E >address
0	0	adder	0	adding	{	==> < F >adding
0	0	adder	0	addition	{	==> < G >addition

cake' instead of 'adder'

col-0	col-1	col-2	col-3	col-4	col-5	
0	0	ant	0	antler	{	==> < A >antler
0	0	ant	0	antivirus	{	==> < B >antivirus
0	0	ant	0	ant hill	{	==> < C >ant hill
0	0	cake	0	L*E*A*F*	{	NOT THIS ONE !

Conclusions about code

Moving forwards one level means some of the right-hand column is copied to make the new left. The obvious question is "How do you know which members DON'T get used? The answer appears in the last block of the previous section called "procedure affects workbench".

From here, we have an idea how the algorithm works:

- Unless the first workbench is a single row, the first rows are generated using something resembling **$this->hardWired_apexFromOrphanSet** in the existing code.
- Each member of the "old right" is checked for the "LEAF" flag. When that DOSNT happen, we generate a row.
- Every row is repeated for each of the one or more "children" from matching that member as currently supplied by the "Spread Along" function shown below.
- There MUST be a point where there are no rows generated - That means the loop is completed.

Spread Along Function

```
32        private function hardWired_spreadAlong($picked){
33            $EOK = $this->EndOfKey;
34            $giveBack=[];
35
36            switch ($picked) {
37                case "a":
38                    $giveBack=array("ant","adder");
39                    break;
40
41                case "b":
42                    $giveBack=array("brick","banjo","bank");
43                    break;
44
45                case "c":
46                    $giveBack=array("cello","catcher","candle",
47                                    "coral","cake");
48                    break;
49
50                case "ant":
51                    $giveBack=array("antler","antivirus","ant hill");
52                    break;
53
54                case "adder":
55                    $giveBack=array("address","adding","addition");
56                    break;
57
58                default:
59                    $giveBack=array($EOK);
60
61            }; // End switch
62          return $giveBack;
63        } // End hardWired_spreadAlong
```

What comes Next

As the project moves to its second stage, development will continue based on three classes:

A. LinoBrick does the work discussed so far; it places content in the correct cells.

B. LinoWall does the work required to join the labels together; it corresponds to the dotted lines in the “Bubble Chart” picture, at the start of this chapter.

C. ToolBox supports functionality that both A and B are able to use.

Although “ToolBox” remains a private class within LinoBrick, it CANT permanently stay that way. Later on, restructuring will make “ToolBox” available to both of the others.