COMP102

Part 1

Data Types and Structures

Lecture 3

Multi-Dimensional Arrays (II)

An Example Application

Giant Characters Revisited

2 drawbacks of encoding characters as sequences of String:

1. The appearance of a ‘character’ was fixed using symbols ’*’ and ’ ’.
2. Print out of sequences of characters was from the top of a page downwards, rather than left-to-right.

Requirements (I)

Define a class, **GIANT_TERNARY**, to hold representations of the three digits **0**, **1** and **2** in a 2-dimensional **char** array.

This class must contain methods to:

1. Represent a symbol using any pair of characters.
2. Print out any row of the 2-dimensional array depicting a given digit.
3. Print out the representation of a digit in an instance.

Requirements (II)

Use this class can be used within a program that converts a given input (non-negative) integer into a **ternary** representation and prints this (from left-to-right) representation using GIANT_TERNARY

The GIANT_TERNARY Class Definition

```
GIANT_TERNARY
-----------------------------------------
protected char fg='*'
protected char bg=' '
protected char[ ][ ][ ] AllDigits =
                    new char [3][9][9]
-----------------------------------------
public GIANT_TERNARY()
public void ChangeOver(char, char)
public void PrintRow(int, int)
public void PrintAll(int,int)
```

An instance of GIANT_TERNARY has 3 instance variables:

char types fg, bg: foreground and background symbols

A 3-D **char** array AllDigits[3][9][9].

The three component (2-dimensional) arrays

AllDigits[0]
AllDigits[1]
AllDigits[2]

store the depiction of the relevant digit (i.e. 0, 1, 2)


```
public class GIANT_TERNARY
  {
  protected char fg='*';
  protected char bg=' ';
  protected char[][][] AllDigits= new char[3][9][9];
  public GIANT_TERNARY()
    {
    final char[][][] TempDigits = {
                 { {bg,bg,bg,fg,fg,fg,bg,bg,bg},
                   {bg,bg,fg,fg,fg,fg,fg,bg,bg},
                   {bg,fg,fg,bg,bg,bg,fg,fg,bg},
                   {bg,fg,fg,bg,bg,bg,fg,fg,bg},
                   {bg,fg,fg,bg,bg,bg,fg,fg,bg},
                   {bg,fg,fg,bg,bg,bg,fg,fg,bg},
                   {bg,fg,fg,bg,bg,bg,fg,fg,bg},
                   {bg,bg,fg,fg,fg,fg,fg,bg,bg},
                   {bg,bg,bg,fg,fg,fg,bg,bg,bg}
                 },
                 { {bg,bg,bg,fg,fg,fg,bg,bg,bg},
                   {bg,bg,fg,bg,fg,fg,bg,bg,bg},
                   {bg,fg,bg,bg,fg,fg,bg,bg,bg},
                   {bg,bg,bg,bg,fg,fg,bg,bg,bg},
                   {bg,bg,bg,bg,fg,fg,bg,bg,bg},
                   {bg,bg,bg,bg,fg,fg,bg,bg,bg},
                   {bg,bg,bg,bg,fg,fg,bg,bg,bg},
                   {bg,fg,fg,fg,fg,fg,fg,fg,bg},
                   {bg,fg,fg,fg,fg,fg,fg,fg,bg}
                 },
                 { {bg,bg,fg,fg,fg,fg,fg,bg,bg},
                   {bg,fg,fg,bg,bg,bg,fg,fg,bg},
                   {bg,fg,fg,bg,bg,bg,fg,fg,bg},
                   {bg,fg,fg,bg,bg,bg,fg,fg,bg},
                   {bg,bg,bg,bg,bg,fg,fg,bg,bg},
                   {bg,bg,bg,bg,fg,fg,bg,bg,bg},
                   {bg,bg,bg,fg,fg,bg,bg,bg,bg},
                   {bg,bg,fg,fg,fg,fg,fg,fg,bg},
                   {bg,fg,fg,fg,fg,fg,fg,fg,bg}
                 }
               };
     AllDigits = TempDigits;
     }
```



```
//
// Change the default, to use
// the foreground symbol (fore) and
// background symbol (back).
//
public void ChangeOver (char fore, char back)
  {
  for (int i=0; i<3; i++)
    for (int j=0; j<9; j++)
      for (int k=0; k<9; k++)
        if (AllDigits[i][j][k]==fg)
          AllDigits[i][j][k]=fore;
        else
          AllDigits[i][j][k]=back;
  fg=fore; bg=back;
  }


//
// Given a digit (0,1, or 2), output the
// specified row (in row_value) of its
// representation as a String, without a
// new line being thrown after printing.
//
public void PrintRow( int row_value, int ternary_digit )
  {
  System.out.print(String.valueOf(
             AllDigits[ternary_digit][row_value]
                                   ));
  }
```



```
//
// Print out the representation of a ternary digit.
//
public void PrintAll (int ternary_digit )
  {
  for (int i=0; i<9; i++)
     {
     PrintRow(i,ternary_digit);
     System.out.println();
     };
  }
```

Requirements (II)

Given user-supplied foreground and background symbols to be used in printing, and a **non-negative** integer, n, output the ternary representation of n with this printed from left to right using the GIANT_TERNARY form.

Analysis

We can view the solution program as involving the stages below:

1. Input foreground and background printing symbols.
2. Create instance of GIANT_TERNARY using these.

3. Read the non-negative integer value.
4. Convert this to an equivalent ternary form.
5. Output the ternary representation.

The first part is (fairly) trivial.

For the second part the Instance method

ChangeOver(**char** fore, **char** back)

provided by the GIANT_TERNARY Class is used.

```
static GIANT_TERNARY NumberForm=new GIANT_TERNARY();
static char fore;
static char back;
...

public static void main( String[] args )
                    throws IOException
  {
  Get_fore_and_back();
  NumberForm.ChangeOver(fore,back);
  }
```

The final item of input is the integer value to be printed in ternary.

This is held in an **int** *InputValue*.

The remaining stages - (4) and (5) - constitute the bulk of the processing needed.

Have to consider,

1. The form the sequence of ternary digits will take.
2. How to construct this from any non-negative integer value.

For (1) there is a choice between using

An array of **int** (to index values 0,1,2)

or

A String.

Advantages of using a String

1. Don't have to woory about how long this is.
2. Easy to avoid 'leading' 0s.

Disadvantage of using a String

Have to translate from a character to an (integer) index in order to find the correct digit to print.

Solution in terms of storing the ternary representation in a String, is used.

String TernaryValue

is the variable used to hold the ternary representation of *InputValue*.

Conversion Process

```
//*************************************************************
// Convert integer to ternary representation returning String *
// of the encoding digits as the result.                      *
//*************************************************************
public static String IntegerToTernary( final int n )
  {
  String res = new String();
  int tn = n;          // Save value of input parameter.
  int next_digit;      // The next ternary digit to be stored
  if (tn==0)           // Can convert 0 immediately.
    {
    res= "0"+res;
    }
  else
    {
    while (tn>0)
      {
      next_digit = tn%3;   // The least significant ternary digit
      tn=tn/3;             // 'shift' off the current value of tr
      //
      // Append the correct symbol to the result String.
      // Could do this directly (i.e. avoiding case statement)
      // by using an integer-character encoding, however, this
      // assumes a specific encoding scheme (ASCII say) and
      // is a bit opaque.
      //
      switch(next_digit)
        {
        case 0: res = "0"+res; break;
        case 1: res = "1"+res; break;
        case 2: res = "2"+res;
        };
      };
    };
  return res;
  }
```


The final stage is the actual output of the data in *TernaryValue* as a sequence of GIANT_TERNARY templates.

This is carried out within *main()*.

GIANT_TERNARY provides an Instance method that outputs a single specified row of the 9-by-9 matrix without throwing a line-break after printing.

print a new_line
Print the first row of each digit in TernaryValue
print a new_line
Print the second row of each digit in TernaryValue
print a new_line
Print the third row of each digit in TernaryValue
print a new_line
...
Print the last row of each digit in TernaryValue
print a new_line

i.e.


```
System.out.println();     // Start on a new line
for (int i=0; i<9; i++)
  {
  for (int j=0; j<TernaryValue.length(); j++)
    {
    //
    switch(TernaryValue.charAt(j))
      {
      case '0': NumberForm.PrintRow(i,0); break;
      case '1': NumberForm.PrintRow(i,1); break;
      case '2': NumberForm.PrintRow(i,2);
      };
    };
    System.out.println();
   };
```

When run on input "#" (foreground) and "," (background) with integer value 11, the following (correct) output results:

```
,,,###,,,,,,###,,,,,#####,,
,,#,##,,,,,#####,,,##,,,##,
,#,,##,,,,##,,,##,,##,,,##,
,,,,##,,,,##,,,##,,##,,,##,
,,,,##,,,,##,,,##,,,,,,##,,
,,,,##,,,,##,,,##,,,,,##,,,
,,,,##,,,,##,,,##,,,,##,,,,
,#######,,,#####,,,,######,
,#######,,,,###,,,,#######,
```


Summary

The constrained multi-dimensional array used in the example application developed here, illustrates several of the points referred to in the previous lecture:

1. The initiation of such structures is handled in a manner similar to that used for 1-dimensional arrays
2. The interpretation of a **k**-dimensional array as an ordered collection of **(k-1)**-dimensional arrays, plays an important role in the analysis.
3. In the realisation of the main output process, viewing a 2-dimensional array as an ordered collection of 1-dimensional arrays, provides a solution to the problem of producing 'left-to-right' printing.