CS2

Theme:

Object Oriented

Programming

in

Java

Handout : Week7

prepared by N S kumar

Preface

Dear friends,

We have crossed the half way mark. Time runs very fast. Does not wait for any body. Trust you have also mastered 50% of the total course.

We will discuss in this week lots of examples on arrays.

FAQ is yet to be ready. I hope to do this during this week.

As always, I look forward to your feedback. I got a few last time. Expect this number to grow exponentially (exponent far far greater than 1!).

All the best.

N S Kumar
Visiting Professor
Dept. Of Computer Science and Engineering
PES Institutions.
Email : ask.kumaradhara@gmail.com
Date : 1st March 2015

Array:

In India, every city (may be, even a small town) would have a road called MG Road. So also, we have an MG Road in Bengaluru. Which road runs parallel to this on its North? Which road runs parallel to this on its South? Which roads cut across MG Road? There is no way to easily guess. These names after sometime will lose their semantics. I am not sure whether the people of the next decade (or two) would know what MG stands for!

In another part of Bengaluru (Hanumanthanagar and Gavipuram areas), the roads are named after great writers in Kannada. A very commendable job. We have Masti Venkatesha Iyengar Road, V Seetharamaih Road and so on. We cannot guess easily which road is adjacent to it.

Come to the place where I stay – Jayanagar. The roads are all numbered. We have roads called main roads (these run North to South) and roads called crosses (these run East to West). They are all numbered. Searching in these areas is definitely easy as we know what would be adjacent to the 13th Main.

We can also sight a few more similar examples. Houses are numbered. Rooms in hotels and colleges are numbered.

In high school mathematics (Geometry), we would create triangles whose vertices would be A, B & C. Later, we are introduced to analytical geometry, where we refer to the points as (x1, y1), (x2, y2), (x3, y3). We use the same name to refer to a number of variables. We change the subscript.

Similar concept in programming is called Array. This concept came into programming through the language Fortran – thanks to Backus.
Let us list and discuss the concept of arrays:

Features:

- An array has a number of elements. A single name refers to a number of elements. We call such variables as collections or containers.
- Normally all the elements of an array are of the same type. We say that they are **homogeneous**.
- The elements of the array are allocated **contiguously** in memory.
- There is no name for each element of the array. We access the element of the array using an expression (Remember an expression has a value). This expression is normally of type int. We call this expression an **index or subscript.**
- The smallest allowable index (leftmost position in an array – assuming that the elements of the array are lead left to right) could vary. It is called the **Lower Bound**. In most of the languages today, the lower bound is 0.
- The time required to access the element of the array does not depend on its position. We say that the arrays support **Random Access**. The word Random is a misnomer – should have been called direct access. We still use the same terminology.
- Normally, size of the array(the number of elements in the array) is fixed at the point of creation.
- We can have arrays of arrays – this is called a Two Dimensional Arrays. These 2D arrays are similar to the concept of matrices in Mathematics. We can also create Three Dimensional arrays or in general Arrays of any number of dimensions.

Now, let us discuss the concept of array in Java.

- In Java, array is a reference type. We can create this anywhere
- The elements of the array will be on the heap. We create that by using the operator new explicitly or implicitly. We will revisit this point later.
- If we assign one array to another, only reference gets copied. This could result in garbage (old array is gone) and aliasing (both refer to the same array).

- Arrays know about their size. There is a read-only attribute called length which gives the array size.
- Comparing with 'C'
    - arrays are not pointers
    - arrays can be assigned.
    - There is no concept of pointer – therefore no concept of accessing elements of the array using pointer arithmetic.
    - C arrays degenerate to a pointer at runtime. C arrays do not know anything about their size at runtime. Java arrays are properly encapsulated.
    - We can meaningfully pass array as an argument to a function, return an array from a function. These are not directly possible in 'C'
    - 2D arrays in Java are actually an array of references to arrays. In 'C', 2D arrays are mapped to a sequential array of linear memory.
    - C arrays are more efficient compared to Java arrays. They use less space compared to Java arrays.
    - Indexing outside the array bounds causes an exception in Java and an undefined behaviour in 'C'. 'C' is efficient – no checking for index out of bounds – but is not as safe as Java.

We will now discuss a few examples of arrays in Java.

Creation of Array:

This example shows how to create an array reference and how to create an array using dynamic allocation. Array elements are always on the heap and will be initialized by default based on the type of the element.

```
//L25/d1/Example1
public class Example1
{
        public static void main(String[] args)
        {
                int[] a; // is an array reference
                // no array at this point
                int b[]; // another array reference
                // cannot specify the size as in 'C'
                // create an array
                a = new int[5];
                // a : reference; refers to an array of 5 int
                // length is an attribute of array; not a method
                // 'C'  is stupid ; C arrays do not know their size
                // all elements created on the heap will have
                //       default value based on the type
                System.out.println(a.length); // 5
                // all zeroes
                for(int i = 0; i < a.length; i++)
                {
                        System.out.print(a[i] + "\t");
                }
                System.out.println();
        }
}
```

A few more examples of Array Creation.

- Create an array by specifying the size
- Create an array – give a list of elements to initialize with – this list follows the new operator.
- Create an array as in 'C'. Will be same as the earlier case.

Please note that we cannot specify the size while creating the array reference.

```
//L25/d1/Example2
// Array Creation
public class Example2
{
	public static void main(String[] args)
	{
		// 1.
		int[] a = new int[5];
		disp(a); // all zeroes
		// 2.
		int [] b = new int[] {1, 2, 3, 4, 5};
		// initialize as the array is created. The compiler will count the
		number of elements. We can not specify.
		disp(b);
		// 3.
		int[] c = {6, 7, 8, 9, 10};
		// This is same as int[] c = new int[] {6, 7, 8, 9, 10};
		disp(c);
		// b & c are created the same way

		// error; cannot specify the size and the elements
		//int [] d = new int[5]{11, 12, 13, 14, 15};

		// ok. but not commonly used; looks like 'C'
		int e[] = {16, 17, 18, 19, 20};
		disp(e);

	}
	public static void disp(int[] x)
	{
		System.out.println("length : " + x.length);
		for(int i = 0; i < x.length; i++)
		{
			System.out.print(x[i] + "\t");
		}
		System.out.println();
	}
}
```

Traversal of an Array:

Visiting each element of a collection is called **traversal**.

There are two ways of accessing each element of the array.

a) Using a for loop, varying the index from 0 till the array length:
   This is similar to the way we traverse an array in 'C'.

b) Ask an array to give the next element each time until the array has no more elements.
   For this, we use enhanced for loop in Java. We have already used this feature while discussing enum classes.

We will discuss the differences between these approaches in the following examples.

The following example illustrates how we can find the sum of elements of an array using both these types of traversal.

```
//L25/d1/Example3.java
// Array traversal
// a) using a for loop
// b) enhanced for loop
//	loop variable is a copy of the element of the array
```

```
public class Example3
{
        public static void main(String[] args)
        {
                int[] a = {10, 20, 30, 40, 50};
                System.out.println("sum : "   + findSum(a));
                System.out.println("sum : "   + findSum1(a));


        }
        // using indexing with a for loop
        public static int findSum(int[] x)
        {
                int s = 0;
                for(int i = 0; i < x.length; i++)
                {
                        s += x[i];
                }
                return s;
        }
        // using enhanced for  loop
        // the loop variable e is a copy of an element of the array
        public static int findSum1(int[] x)
        {
                int s = 0;
                for(int e : x)
                {
                        s += e;
                }
                return s;
        }
}
```

Let us consider an array which is storing the marks of all students in CS2 T1. Let us say that our college decides to modify the marks of every student by adding some grace marks(Or by subtracting some marks gracefully!). Can we use any of these looping structures to achieve the expected result? Think about this before you read what follows.

```
//L25/d1/Example4.java
// Array traversal
//      Array is a reference type
//  parameter is a copy of the array reference
//      if the reference is changed, nothing happens to the referent
//  if the referent is changed through the reference, that change will remain.

public class Example4
{
        public static void main(String[] args)
        {
                int[] a = {10, 20, 30, 40, 50};
                addOne(a);
                disp(a);

        }
        public static void addOne(int[] x)
        {
                for(int i = 0; i < x.length; i++)
                {
                        // would this modify the elements of the array?
                        x[i] = x[i] + 1;
                }

        }
```

```
	public static void disp(int[] x)
	{
		System.out.println("length : " + x.length);
		for(int i = 0; i < x.length; i++)
		{
			System.out.print(x[i] + "\t");
		}
		System.out.println();
	}
}
```

In the program given above, you will find that the array has changed.

```
//L25/d1/Example5.java
// Array traversal
//	loop variable is a copy of the element of the array
//  this copy is of primitive type
//  changing this will not affect the array

public class Example5
{
	public static void main(String[] args)
	{
		int[] a = {10, 20, 30, 40, 50};
		addOne(a);
		disp(a);


	}
	// this will not work
	public static void addOne(int[] x)
	{
		for(int e : x)
		{
			// e is an element of the array ; a copy of it
			// does modifying e affect the array?
			e = e + 1;
		}


	}
	public static void disp(int[] x)
	{
		System.out.println("length : " + x.length);
		for(int i = 0; i < x.length; i++)
		{
			System.out.print(x[i] + "\t");
		}
		System.out.println();
	}
}
```

In the example cited above, the array remains unchanged.

Let us summarize the differences between these two ways of traversal.

- **In case of the enhanced for loop, we have to walk through the whole collection. We cannot limit the traversal to part of an array.**
- **In case of the enhanced for loop, we cannot modify the elements of the array.**

Guideline: Prefer enhanced for loop whenever you can.

Function returning an array:

C arrays degenerate to a pointer at runtime – amnesia sets into the array at runtime. Java arrays remain arrays at runtime as well. So, we can write methods in Java to create arrays and return them. This is a simple example to operate on each element of the array (we are squaring each element here) and put the resultant element into an array and return the array as the result.

```
//L26/d1/Example1.java
// function returning an array
// Array traversal
public class Example1
{
	public static void main(String[] args)
	{
		int[] a = {10, 20, 30, 40, 50};
		// observe that we should not allocate for b
		// if we do, after the assignment, the array to which refers will become
garbage
		int[] b = computeSquares(a);
		disp(a);
		disp(b);

	}
	public static void disp(int[] x)
	{
		for(int e : x)
		{
			System.out.print(e + "\t");
		}
		System.out.println();
	}
	public static int[] computeSquares(int[] x)
	{
		// observe the necessity of allocation
		// observe how the size is fixed here
		int[] res = new int[x.length];
		for(int i = 0; i < x.length; i++)
		{
			res[i] = x[i] * x[i];
		}
		return res;
	}
}
```

There is one catch, We should be able to figure out the number of elements in the resultant array at the point of creation. This example was simple as the number of elements in the resultant array is same as the number of elements in the parameter array.

The example given below indicates a way of solving this problem. We are trying to find the elements which are even in the given array. The size of the resultant array can at most be same as the size of the given array. So, we create an array for the worst case, then fill it up based on our logic(we use a nice term here- we say we populate the array!). Once the processing of the input array is complete, we know the size of the resultant array. We now create the resultant array of the required size, copy the elements and then return the array.

```
//L26/d1/Example2.java
// Array traversal
// function returning an array
public class Example2
{
        public static void main(String[] args)
        {
                int[] a = {1, 2, 3, 4, 5};
                int[] b = findEvens(a);
                disp(a);
                disp(b);
        }
        public static void disp(int[] x)
        {
                for(int e : x)
                {
                        System.out.print(e + "\t");
                }
                System.out.println();
        }
```

```
    // size of the result is not determinable before hand
    public static int[] findEvens(int[] x)
    {
        // create an array as big as x
        int[] temp = new int[x.length]; // a BIG array for the worst case
        int j = 0;
        for(int i = 0; i < x.length; i++)
        {
            if(x[i] % 2 == 0)
            {
                temp[j++] = x[i];
            }
        }
        // # of elem in temp is j
        // create an array of size j
        int[] res = new int[j];
        for(int i = 0; i < j; i++)
        {
            res[i] = temp[i];
        }
        return res;
    }
}
```

Searching and Sorting:

I spend most of my time looking for a thing – a bank document, a sheet of paper on which I wrote the questions for T2. Somehow this phrase “place for a thing and thing for a place” has never been for me. I feel things around me do not follow Newton. They seem to move around on their own to put me into trouble. So I always keep **searching**. If I could arrange them a bit better – **sort** them, then I can save a few precious minutes of my life – which I could then use for better things in life – like watching a movie.

Sorting and Searching are two very important activities in Computer Programming. There is a book on this topic which has more than 700 pages (759 pages to be exact). **The Art of Computer Programming: Volume 3: Sorting and Searching : Donald E Knuth.**

Searching : We are looking for an element in an array which satisfies a given condition – more often than not – check whether a given element exists in an array. In our examples, we will search the array from left to right and check whether the given element exists.

There are a few interesting questions about the design of this method.

1. Should the method display whether the element is found or not in the method itself?

NO. A method should display only if it is meant for it. If the method displays whether the element is found or not, what can the user of this method do with it? Would you be happy if you order for dosa in Vidyarthi Bhavan and the cooks make dosa and then tell you that dosa was made – (they do not give it to you)?

2. Should the method return a boolean value indicating the element was found or not found?

This is definitely better than the earlier method. Let us consider a case where you ask the librarian to find a book for you(surprising?). He checks the book shelf for your book and then tells you that the book is there in the book shelf and he does not tell you where it is. Is that helpful?

3. Should the method return the value found?

This could be useful sometime but not when we are looking for an element in the array – we know the element already. There is another problem. What should the method return if the element is not found? 0 or 100 or … ? This could be the element we are looking for!

4. Should the method return the position where it is found?

Yes if the search is in an array. Then what should the method return if the element is not found. It should be some index which is not valid for this search. It could be -1 if we are searching an array. If we are searching in some other collection, we require some other mechanism. We will discuss that mechanism towards the end of this course.

In these search algorithm, we may have to break out of the loop if the element we are looking for is found. There are a number of ways of achieving this.

a) use **return <expr>;** statement.

b) use **break;** statement.

Both these are not considered good. It is difficult to read a method with multiple returns. It is also difficult to follow a loop if it has break and continue.

Good Guidelines :

A method should have a single return.

Use break only if switch.

never continue.

c) exit the loop by using a conditional variable.

This loop variable should be a boolean and the name should indicate what is stands for. **It should not be called flag.** The name flag is a terrible name and has been misused a lot in programming. Many times this flag would take any one of three values!

```
//L26/d1/Example3.java
// searching
import java.util.*;
public class Example3
{
	public static void main(String[] args)
	{
		int[] a = {1, 2, 3, 4, 5};
		Scanner scanner = new Scanner(System.in);
		int e = scanner.nextInt();
		int pos = findElement(a, e);
		if(pos == -1)
		{
			System.out.println("not found");
		}
		else
		{
			System.out.println("found at pos : " + pos);
		}

	}

/*
	public static int findElement(int[] x, int e)
	{
		boolean found = false;
		int i; // why here and not in the loop
		for(i = 0; ! found && i < x.length; i++)
		{
			if(x[i] == e)
			{
				found = true;
```

```
                }
            }
            if(found)
            {
                // these are fairly common errors – called off-by-one errors.
                return i; // wrong; should be i - 1
            }
            else
            {
                return -1;
            }
        }
*/
//      better way of writing
        public static int findElement(int[] x, int e)
        {
            boolean found = false; // default initialization; not found so far
            int pos = -1; // not found so far
            int i;
            for(i = 0; ! found && i < x.length; i++)
            {
                if(x[i] == e)
                {
                    found = true;
                    pos = i;
                }
            }
            return pos;
        }
}
```

You may observe how nicely found and pos have been initialized. They indicate the state of the search at that point. These are changed if the element is found.

Sorting: Arranging elements in order based on some condition is sorting. We will have a look at a very simple example of arranging elements in non decreasing order. This algorithm has an interesting name – is called BubbleSort. According to some Computer Scientists, the only interesting thing about this algorithm is its name!

Let us take this sample array and understand how BubbleSort works.

We compare the adjacent elements. If the element on the left is greater than the element on the right, we interchange the elements (we swap them). By the time we traverse all the elements, the biggest element would have bubbled out to the last position. Now the problem size has decreased by 1. We have to sort the size of the array – 1 elements. If we repeat this size of the array – 1 times, all but the first would have been in the right position. The first one has no choice – has to be at the right position.

```
22 55 44 11 33
^   ^   ok
    ^   ^  swap
22  44   55 11 33
        ^  ^     swap
 22 44  11  55 33
           ^  ^  swap
  22 44 11 33 55
```

55 is in its right position.
You can repeat this for the following elements.
22 44 11 33.

The most difficult part in these algorithms is in deciding that bounds of the loops. We should try to associate some meaning for the loop variable.

```
for(int i = 0; i < l - 1; i++) …
```

In our example, the outer loop uses the variable i to vary from 0 till size of the array – 1. What does i stand for? How do we arrive at the condition of the loop?

We can look upon i as indicating the number of elements sorted so far. Our requirement is sort l – 1 elements. Once we do that, the first element will be in its right place. So, we can now appreciate this loop.

What does the inner loop indicate?

```
for(int j = 0; j < l - 1 - i; j++) …
```

By now, the last i elements have been sorted. We are required to compare adjacent elements until the right most element is l – i - 1.

If we go wrong in these bounds, the program may not work for some cases. We may be lucky – in that case we get runtime exception in Java and segment fault in 'C'. But unfortunately we are not always so lucky. The program will work at home and will not work in the lab exam!

You should be extremely careful about the bounds in the looping constructs. Analyze what the variables stand for with respect to the problem being solved.

```
//L26/d1/Example4.java
// sorting
import java.util.*;
public class Example4
{
        public static void main(String[] args)
        {
                int[] a = {22, 55, 44, 11, 33};
                disp(a);
                mysort(a);
                disp(a);
        }
        // disp method not shown in this example
        // bubble sort
        public static void mysort(int[] a)
        {
                int l = a.length;
                for(int i = 0; i < l - 1; i++)
                {
                        for(int j = 0; j < l - 1 - i; j++)
                        {
                                if(a[j + 1] < a[j])
                                {
                                        int temp = a[j];
                                        a[j] = a[j + 1];
                                        a[j + 1] = temp;
                                }
                        }
                }
        }
}
```

Library Methods:

There were days when we would do everything at home. Our ancestors would do lots of hard work – pounding rice at home, powdering sugar, making nice sweets. But we are a lazy lot. We buy sweets in the shops. Let somebody do the hardwork. We want to use our time for whatever we consider better – chat on fb, comment on twitter, sms on whatsapp, … (I am sure you will have a very long list here).

In programming, there are already methods available which we can directly use. We call these helpful collections as libraries.. We do not want to reinvent every wheel. This helps us in many ways.

When we want to build a house, most probably we will not fell a tree ourselves, cut that into lots and make doors out of them. We may get planks already made and combine them to make doors.

Let us understand why we should use libraries.

- These methods are written by experts. These would be most probably better than anything we write.
- These methods would have been tested. These would work correctly. So, we can avoid bugs in our programs.
- Since these routines are already available, we can use them and build something big easily.

Java provides a class called java.util.Array which has lots of useful methods. Please check this in java documentation.

In this class, all methods are static. We do not invoke these methods using an object. We will pass an array as argument to these methods,

Remember that we do not have functions out side of a class as we can do in 'C'. This is the workaround in Java. Such a class with only static methods is called a Utility class.

Some of the useful methods are:

a) Arrays.toString(array)

converts the array to a String.

b) Arrays.copyOf()

returns an array after copying – we could have used in the example of finding even elements in an array.

c) Arrays.fill()

fills the array with the given elemens

d) Arrays.sort()

Arrange elements in order. We wasted our time and energy in writing our own bubblesort!

e) Arrays.binarySearch()

Search an element in a sorted array.

You may want to experiment with these methods.

```
//L27/Example1.java
// check the builtin class called Array
// all methods are static methods
// such a class is not exactly object oriented
// we call them utility classes
// another example : Math class
import java.util.*;
public class Example1
{
	public static void main(String[] args)
	{
		int[] a = {33, 99, 11, 55, 88, 44, 66, 22, 77 };
		System.out.println(Arrays.toString(a));
		Arrays.sort(a);
		System.out.println(Arrays.toString(a));
```

```
            int[] b; // what if we allocate here? what will
            // happen to the size after copying?
            // compare the outputs
            b = Arrays.copyOf(a, 5);
            System.out.println(Arrays.toString(b));
            b = Arrays.copyOf(a, 15);
            System.out.println(Arrays.toString(b));

            int[] c = {1, 2, 3, 4};
            Arrays.fill(c, 100);
            System.out.println(Arrays.toString(c));

            int[] d = {1, 2, 3, 4};
            Arrays.fill(d, 0, 2, 200); // 0 to 2 and 2 not inclusive
            System.out.println(Arrays.toString(d));

            //binary search
            System.out.println("search : " +
                    Arrays.binarySearch(a, 77));
            System.out.println("search : " +
                    Arrays.binarySearch(a, 82));
            // find the meaning of the return value if search fails.
      }
}
```

MultiDimensional Array:

A matrix has two dimensions – has rows and columns. We talk about n dimensional vector spaces. How do we represent them in our programs. Java allows us to create multidimensional arrays. This is an example of 2D arrays.

Let us understand how Java creates these two dimensional arrays.

```
            int a[][] = {{1, 2, 3}, {4, 5, 6, 7},
                        {8, 9, 10}};
```

a : is a array reference which can refer to an array of references to arrays.
In this example, a will be refer to 3 elements each of which refers to an array of 3 elements. This way of arrangement is called **row pointer arrangement**. You may want to find out how 'C' does. It is totally different.

We can walk through an array using any of the two methods we discussed earlier. (No amnesia?).

```
//L28/Example1.java
//2D array
public class Example1
{
	public static void main(String[] args)
	{
		int a[][] = {{1, 2, 3}, {4, 5, 6, 7},  {8, 9, 10}};
		disp(a);
		disp1(a);
	}
	// storage : row pointer way of storage
	public static void disp(int[][] x)
	{
		// x is an array reference
		int i; int j;
		for(i = 0; i < x.length; i++)
		{
			// x[i] is also an array reference
			for(j = 0; j < x[i].length; j++)
			{
				System.out.print(x[i][j] + "\t");
			}
			System.out.println();
		}
	}
```

```
    public static void disp1(int[][] x)
    {
        for(int[] r : x) // r : reference to a row
        {
            for(int e : r)
            {
                System.out.print(e + "\t");
            }
            System.out.println();
        }
    }
}
```

This example indicates exactly how a 2D array is created.

```
//L28/Example2.java
// 2 D array
public class Example2
{
    public static void main(String[] args)
    {
    // observe how 2D arrays are created
    /*
        int a[][];
        a = new int[][]{{1, 2, 3}, {4, 5, 6, 7},
                        {8, 9, 10}};
        disp(a);
        disp1(a);
    */
        int a[][];
        a = new int[3][];
        a[0] = new int[]{1, 2, 3};
        a[1] = new int[]{4, 5, 6, 7};
        a[2] = new int[]{8, 9, 10};
```

```
		disp(a);
		disp1(a);

	}
	public static void disp(int[][] x)
	{
		int i; int j;
		for(i = 0; i < x.length; i++)
		{
			for(j = 0; j < x[i].length; j++)
			{
				System.out.print(x[i][j] + "\t");
			}
			System.out.println();
		}
	}
	public static void disp1(int[][] x)
	{
		for(int[] r : x)
		{
			for(int e : r)
			{
				System.out.print(e + "\t");
			}
			System.out.println();
		}
	}
}
```

There are a couple of interesting (to me!) examples of use of 2D array.
The first one generates a magic square. Add the numbers rowwise, columnwise or diagonally, the sum is same. This is supposed to be Indian Contribution to the field of popular mathematics.

The second one generates Pascal Triangle – these are co-efficients of binomial expansion.

In the first example, the array is rectangular. The number of columns in each row is same. In the second example, the number of columns in each row is different. Such a 2D array is called a jagged array.
Walk through the two programs and understand the logic.

```
//L28/Example3.java

// magic square : order of the square matrix should be odd
public class Example3
{
	public static void main(String[] args)
	{
		int n = 5;
		int sq[][] = new int[n][n];
		// all elements are zero as per the java rule
		//	of initializing variables on the heap
		//	with default value based on the type
		fill(sq);
		disp(sq);
	}
```

```
public static void disp(int[][] x)
{
        int i; int j;
        for(i = 0; i < x.length; i++)
        {
                for(j = 0; j < x[i].length; j++)
                {
                        System.out.print(x[i][j] + "\t");
                }
                System.out.println();
        }
}
// rules:
// put 1 in top row mid col
// repeat for numbers from 2 to n * n
        // move up and to right each time
        // if out of diagonal move 2 Down 1 Left
        // else if out of row goto the last row
        // else if out of col goto the first col
        // if filled up, move 2 Down 1 Left
        // fill the square with the next number
        public static void fill(int[][] sq)
        {
                int n = sq.length;
                int i = 0; // zeroth row
                int j = n / 2; // mid col
                int k = 1;
                sq[i][j] = k;
                for(k = 2; k <= n * n; k++)
                {
                        i--; j++;
                        if(i < 0 && j == n)
                        {
                                i += 2; j--;
                        }
                        else if(i < 0)
                        {
                                i = n - 1;
                        }
                        else if(j == n)
                        {
                                j = 0;
                        }
                        if(sq[i][j] != 0)
                        {
                                i += 2; j--;
                        }
                        sq[i][j] = k;
                }
        }
}
```

```
//L28/Example4.java
// Pascal Triangle
// n = 5
// output expected:
//                 1
//             1    1
//           1   2    1
//         1   3    3    1
//       1   4   6    6    1
//    1    5  10   10    5    1

// data structure:
// jagged array
// 0th row has 1 elem
// 1st row has 2 elem
// so on

public class Example4
{
        public static void main(String[] args)
        {
                int n = 5;
                // n + 1 rows
                // display  the structure after this loop
                int[][] pt = new int[n + 1][]; // creates n + 1 array references
                for(int i = 0; i <= n; ++i)
                {
                        pt[i]  =  new  int[i  +  1];  //  creates  an  array  –  each  time  of  a
different size
                }
                fill(pt);
        disp(pt);
}
public static void disp(int[][] x)
{
        int i; int j;
        int n = x.length - 1;
        for(i = 0; i <= n; i++)
        {
                int m = x[i].length;
                // display some white spaces in the beginning
                for(j = 0; j <= (n - m); j++)
                {
                        System.out.printf("   ");
                }
                for(j = 0; j < m; j++)
                {
                        // this is similar to printf of 'C'
                        System.out.printf("%5d ",x[i][j]);
                }
                System.out.println();
        }
}
```

```
    // follow the logic for a small value of the size.
    public static void fill(int[][] pt)
    {
        int n = pt.length - 1;
        for(int i = 0; i <= n; i++)
        {
            pt[i][0] = pt[i][i] = 1;
            for(int j = 1; j < i; j++)
            {
                pt[i][j] = pt[i-1][j-1] + pt[i-1][j];
            }
        }
    }
}
```

I have put in my work for this week, It is your turn to put in effort.
A deer will not enter the mouth of a sleeping lion – it has to do its work – hunt.

Nahi suptasya simhasya pravishanti mukhe mrigaha!

Exercises:
Write methods for the following

1. find the biggest element in an array
2. find the second biggest element in an array
3. find the difference between successive elements in an array and return the result as an array. It is like finding the partnership given the fall of wickets.
4. Find the cumulative sum. Return an array. It is like given the partnerships for each wicket, find the fall of wickets.
5. Check whether the elements of an array are in ascending order
6. Check whether every element of an array is odd
7. check whether some element of an array is odd
8. reverse a given array.
9. Add two matrices
10. multiply two matrices.

Enjoy these assignments until we have some more. All the best.