EEL 4832
ENGINEERING DATA STRUCTURES

INTRODUCTION TO JAVA

First program:

```
// First program
//  5/15/06

public class FirstProgram
{
    public static void main( String [ ] args )
    {
        System.out.println("Engineering Data Structures" );
    }
}
```

To compile this program and run on Olympus

```
>javac FirstProgram.java

>java FirstProgram

       Engineering Data Structures
```

Primitive types:

int 32 bit integer;
long 64-bit integer;
float , double
char
Boolean

A program to print text and data on the same line :

```
  public class p2
  {
     // Program to print values

      public static void main( String [ ] args )
      {
          int a = 12, b = 8, c = 6;

          System.out.println("a is "+ a);
          System.out.println("a+b is " +a+b );
          System.out.println(" real a+b is "+(a+b)  );
          System.out.println(a+b  " is also a+b );

          a = c;
          System.out.println(a + " " + b + " " +  c );
      }
  }
```

output of the program:

```
a is 12
a+b is 128
real a+b is 20
20 is also a+b
a+b+c is 1286
6 8 6
```

Type conversions:

```
int  div;
float result1, result2;
int x = 8;
int y = 12;
div = x/y;
result1 = x/y;
result2 = (float)x/y;
```

Output:

```
div = 0,
result1 = 0.0,
result2 = 0.75
```

Methods:

A function in Java is called a method

```
public class MinTest
{
    public static void main( String [ ] args )
    {
        int a = 3;
        int b = 7;

        System.out.println( min( a, b ) );
    }

    // Function definition
    public static int min( int x, int y )
    {
        return x < y ? x : y;
    }
}
```

Overloading of method names:

Java allows several methods to have same name , as long as their parameter list differs.

```
public class p2
{
    // Program to find minimum value

    public static void main( String [ ] args )
    {
        int a = 12, b = 8, c = 6;

        System.out.println("min a b is "+min(a,b) );
     System.out.println("min a,b,c is"+min(a,b,c) );

        }

    public static int min( int x, int y )
    {
        return x < y ? x : y;
    }

    public static int min( int x, int y, int z )
    {
        int minimum;
        minimum= x < y ? x : y;
        minimum= minimum < z ? minimum : z;
        return minimum;

    }
```

Strings:

Strings are handled using pointers in C. In Java the corresponding type *(reference type)* is used for handling strings. As in C, Operators such as <, >, >= are not allowed in Java. Two Strings lhs and rhs can be compared using `lhs.compareTo(rhs)`
This will return a negative number, zero, or a positive number depending on whether lhs is lexicographically less than, equal to or greater than rhs.
The only operator allowed is '+', which is used to concatenate two strings. Other methods used on strings are:

```
String sample = "hello";
int len = sample.length( );              //len is 5
char ch = sample.charAt(1);              //ch is 'e'
String sub = sample.substring(2,5);      //sub is "llo"
```

Arrays:

```
int [] array1;
array1 = new int [100];

int [] array2 = new int [ 100];

int [] array3 = {2, 4, 9, 28, 80};
```

Here is an example of using an array. The program generates random numbers in the range 1 to 100. The number of times a particular value is generated is stored in the corresponding position in the array.

```
import java.util.*;

public class RandomNumbers
{
    // Generate random numbers (from 1-100)
    // Print number of occurrences of each number

    public static final int RANGE     =     100;
    public static final int TOTAL_NUMBERS    = 1000000;

    public static void main( String [ ] args )
    {
        int [ ] numbers = new int [ RANGE + 1 ];
        for( int i = 0; i < numbers.length; i++ )
            numbers[ i ] = 0;

        Random r = new Random( );

        // Generate random numbers in the range 1-RANGE
        for( int i = 0; i < TOTAL_NUMBERS; i++ )
            numbers[ r.nextInt( RANGE ) + 1 ]++;

        // Output the summary
        for( int i = 1; i <= RANGE; i++ )
            System.out.println( i + ": " + numbers[ i ] );
    }
}
```

Here is the sample output of this program:

```
1: 9952
2: 9968
3: 10185
4: 10093
5: 10025
6: 9866
```

Calling arrays in functions in Java is similar to doing it in C.

Dynamic array expansion:

It is possible to define a new Array of any size in Java. Suppose we start with an allocation for 10 integers:

```
int [] arr = new int [10];
```

Suppose later we feel the need to have a slightly bigger sized array. Then we have to copy the elements from the original array.

```
int [] original = arr; //both point to same array
arr = new int [12];
for ( int i=0; i<10; i++)
arr[i] = original[i];
original = null;
```

The process needs to be repeated if after sometime again there is a need to increase the size. A better way would be to double the size of the array whenever requirement exceeds current size. An application of this technique is to read in elements into an array, where the number of elements is not known in advance. Here is an example which reads in strings of unknown lengths.

```
import java.io.*;

public class ReadStrings
{
    public static void main( String [ ] args )
    {
        String [ ] array = getStrings( );
        for( int i = 0; i < array.length; i++ )
            System.out.println( array[ i ] );
    }

    public static String [ ] getStrings( )
    {
        BufferedReader in = new BufferedReader( new
      InputStreamReader( System.in ) );
        String [ ] array = new String[ 5 ];
        int itemsRead = 0;
        String oneLine;

        System.out.println("Enter one string per line; " );
        System.out.println( "Enter empty line at end: " );
```

```
        try
        {
            while( ( oneLine = in.readLine( ) ) != null &&
        !oneLine.equals( "" ) )
            {
                if( itemsRead == array.length )
                    array=resize(array, array.length * 2 );
                array[ itemsRead++ ] = oneLine;
            }
        }
        catch( IOException e )
        {
            System.out.println("IOexception-read abort");
        }

        System.out.println( "Done reading" );
        return resize( array, itemsRead );
    }

    // Resize a String[ ] array; return new array
    public static String [ ] resize(String [] array,
                                    int newSize )
    {
        String [ ] original = array;
        int numToCopy = Math.min(original.length, newSize);

        array = new String[ newSize ];
        for( int i = 0; i < numToCopy; i++ )
            array[ i ] = original[ i ];
        return array;
    }
}
```

Enhanced for loop:

It is possible to access each element in an array through a simple command. To print out the elements of an array D containing strings, we can write:

```
For(String val : D)
     System.out.println(val);
```

ARRAY LIST:

Java provides a built-in function to carry out dynamical array expansion:

```
import java.io.*;

public class ReadStringsWithStringArrayList
{
    public static void main( String [ ] args )
    {
        StringArrayList array = getStrings( );
        for( int i = 0; i < array.size( ); i++ )
            System.out.println( array.get( i ) );
    }

        public static StringArrayList getStrings( )
    {
        BufferedReader in = new BufferedReader( new
      InputStreamReader( System.in ) );

        StringArrayList array = new StringArrayList( );
        String oneLine;

        System.out.println("Enter strings, one/ line; " );
        System.out.println( "Last line is blank line: " );

        try
        {
                while( ( oneLine = in.readLine( ) ) != null
                           &&      !oneLine.equals( "" ) )
                 array.add( oneLine );
        }
        catch( IOException e )
        {
            System.out.println( "Unexpected IO Exception
has shortened amount read" );
        }

        System.out.println( "Done reading" );
        return array;
    }
}
```

Exception Handling:

Handling errors in Java is much more systematic than in C. If you anticipate an error in data value/format/ range, you can make that method to throw an exception, and write another code to catch that exception and generate suitable printout to inform the user about the encountered error. Code that might result in the exception is enclosed in a **try** block, followed by one or more **catch** blocks. Here is an example. In this code the user is asked to enter an integer x, and the code prints out x/2. However, if the user enters characters other than integers, the program throws out an exception.

```
import java.io.*;
public class DivideByTwo
{
    public static void main( String [ ] args )
    {
      //Getting data from the keyboard
       BufferedReader in = new BufferedReader( new
                           InputStreamReader( System.in ) );
        int x;
        String oneLine;

        System.out.println( "Enter an integer: " );
        try
        {
            oneLine = in.readLine( );
            x = Integer.parseInt( oneLine );
            System.out.println( "Half of x is " + (x/2 ) );
        }

        catch( IOException e )
          {
             System.out.println( e );
          }

        catch( NumberFormatException e )
          {
             System.out.println( e );
          }
    }
}
```