Chapter 3

Decision Making and Loop

Dulal Acharjee

In any programming, decision making is one of the important jobs. In JAVA, these statements are available as: if-else, if-elseif-else, case-switch, ternary operators, logical testing of left and right sides etc. These statements are used to test two variables for testing of its values whether larger, smaller or equal. In one word, it is a testing of comparison between two values where, values of variables can be of any data types. The basic rule of comparing is that for each comparison separate decision structure to be built as shown in the syntax shown below, in left side there are two decision structures and in right side there are three decision blocks.

Decision making using if ..else:

Taking decision on different cases is the most important work in programming.
Three models of Syntax of *if* statements are shown below:

Table 3.1 different syntaxes of if—else statements

if(condition)
{
Statement;
}
	if(condition)
	{
	….
	}
	else
	{
	….
	}
		if(condition) {
		….
		}
		else if {
		….
		}
		else{
		….
		}

In this structure of testing, if the *condition* is true, then it will enter within the {..} and will execute statements written within {}, but, if the *condition* is false then it will enter within the structure of ***else{..}*** and will execute the lines of statements mentioned there. Say, *x= 10* and *y= 5*, then if we look at the decision block, *if(10 > 5)* which generates the condition is 'true' as *if (true)*, then control will move towards inside of {..}. Again if we say x=5 and y=10, then, *if (5>10)*, it is *false*, so *if(false)* then control will flow towards else {..} and any statements present within else{..} will be executed. A block diagram of Flow Chart is shown below.

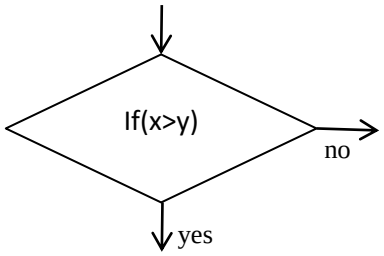

Fig.3.1 a decision box showing inside-condition and two output directions yes, no.

Again, if comparisons are required to do among more than two numbers, then more decision structures to be developed using 'else if'.
Different operators used for testing similarities are:
Greaten than: >
Less than : <
Equal: = =
Not equal : ! =
Less than or equal: < =
Greater than or equal: > =

Problem: wap to demonstrate if..else using some comparison operators.

```
//Demoif.java: to demonstrate if ..else
public class Demoif
{
   public static void main(String[] st)
   {
          int x=10, y=5;
          if(x==y)
         {
             System.out.println("two numbers are equal..");
          }
          Else
         {
             System.out.printf("check  for x= %d and y= %d %n", x,y);
          }
          byte p=55, q=44;
          if(p<q)  { System.out.println(" p is less than q value .");}
          else{System.out.println(" may be p is greater or equal to q"); }
   }
}
OUTPUT:
check  for x= 10 and y= 5
may be p is greater or equal to q
```

Discussions: in this example, different forms of uses of *if..else* are demonstrated. First, if command is shown in the same line as {..}, in next line, printf is used with control identifiers as %d to display two values of x and y. The third on is %n is used for a new line.

Note: if the execution statement is only one line, then no need to write within {}, but, if these are more than one lines then {..} should be used to mark as start and end of the structure.
Lab Test: in machine test students should use other comparison operators and check the values. Also they should learn about three structures as: if{..} else if{…} else{…}

Problem: wap to input three numbers from the keyboard and then find the largest number.

```
//Big3.java: to find largest number among the three.
import java.util.Scanner;
public class Big3
{
   public static void main(String[] st)
  {
          int n1, n2, n3; //for three numbers
          //Scanner ob;
          //ob = new Scanner(System.in);
          Scanner ob=new Scanner(System.in);
          System.out.print("Input first no: ");
          n1 = ob.nextInt();
          System.out.print("Input second no: ");
          n2 = ob.nextInt();
          System.out.print("Input third no: ");
          n3 = ob.nextInt();
          if(n1 > n2 && n1 > n3)
          {
                  System.out.println("Largest number is:"+n1);
          }
          else if(n2 > n3)
          {
                  System.out.println("Largest number is:"+n2);
          }
          else
          {
                  System.out.println("Largest number is:"+n3);
          }
    }
}
```

OUTPUT:

```
Input first no: 8
Input second no: 5
Input third no: 2
Largest number is:8
```

Discussion: In this example, for input of numbers from keyboard, input stream class 'Scanner' is used and there two lines are important for understading of input stream:

```
Scanner ob;
ob = new Scanner(System.in);
```

Here ‘ob’ is an object created from the built in class ‘Scanner’. If we want to work with the object we need to link that object with a memory address of the computer which is performed by the statement:

```
ob = new Scanner(System.in);
```

Another point is that ‘System.in’ is the argument of the constructor ‘Scanner()’. The method ‘nextInt()’ provide facility to read an integer number from input stream(Keyboard) and is a member method of the class ‘Scanner’. These concepts will be more discussed in the chapter of Object Oriented Programming.

Decision Making using switch and case:

Generally, for decision making, three types of statements of if ..else are used as shoen in Table 3.1. The advanced and efficient form of decision making statements are ‘switch..case..’. The method ‘switch(argument)’ may have *int*, *short*, *byte*, *long* , *string* and *char* data types. Also, the statement ‘*case*’ may have both numerical and character type of arguments. Some styles of using ‘switch’ and ‘case’ are shown below:

```
1.    switch(argument){
2.       //Case statements
3.       case 1: {
4.                System.out.println("10");
5.                break;
6.                }
7.       case 2:{
8.               System.out.println("20");
9.               break;
10.              }
11.      case 3:{
12.               System.out.println("30");
13.               break;
14.               }
15.      //if argument is not 1or 2 or 3
16.      default:System.out.println("out of options");
17.   }//end of switch
```

```
//switch2.java: this program shows uses of switch statement as substitute of if..else.
//it is used for options selection.
//The class 'DataInputStream' is for data input operations.
//readLine() input as string, then 'parseInt()' converts string to integer.
//The break statement breaks the structure and come out from the switch structure.
import java.io.DataInputStream;
class switch2
{
    public static void main(String args[])
   {
          DataInputStream in= new DataInputStream(System.in);
```

```
        int x =0;
        String n;
        try
        {
            System.out.println("Input any option(1-3) ");
            n = in.readLine();
            x = Integer.parseInt(n);
            switch(x)
           {
                  case 1:  System.out.println("Do any job1 ");break;
                  case 2: System.out.println("Do any job2 ");break;
                  case 3: System.out.println("Do any job3 ");break;
                  default: System.out.println(" Invalid choice ");
           }
        }
        catch(Exception ee){}   /* Exception or Error handling */
    }
}
```

OUTPUT

```
E:\example>java switch2
Input any option(1-3)
3
Do any job3

E:\example>java switch2
Input any option(1-3)
1
Do any job1
```

The **switch(argument)** statement may have string or char type of arguments, i.e. the argument will be checked by 'case' statement. If argument of string type, then 'case' will check different options like:

```
1. case "add": ans = x+y;
2. break;
3. case "sub": ans = x-y;
4. break;
5. default: ans = 0;
```

In the next example, read() method is used which can read upto one byte of data, so, it is required to convert it to character type of data. In this program the variable name is 'st' which will hold the option selections as 'a' or 'b' or 'c' etc. Sometimes users may press capital letters also, to hold capital letters, case options are designed as case 'a': case 'A' , case 'b': case 'B' … in these ways.

```
//switch3.java: this program shows uses of switch statement as substitute of if..else.
// it is used for options selection. Here, inputs are character
// type of data which can input single character. 'switch(st)' can take an argument
//as character type.
import java.io.DataInputStream;
```

```
class switch3
{
    public static void main(String args[])
   {
          char st;
          try
          {
              System.out.println("Input any option(A/B/C) ");
              st = (char)System.in.read();
              switch(st)
             {
                    case 'a':
                    case 'A':
                    {
                        System.out.println("Do any job1 of this optin ");
                         break;
                     }
                    case 'b':
                    case 'B':
                    {
                        System.out.println("Do any job2 ");
                        break;
                     }
                     case 'c':
                     case 'C':
                     {
                         System.out.println("Do any job3 ");
                         break;
                      }
                      default: System.out.println(" Invalid choice ");
               }//end of switch
          }//end of try
          catch(Exception ee){}   /* Exception or Error handling */
     }//end of main
}//end of class
```

OUTPUT:

```
E:\example>java switch3
Input any option(A/B/C)

Do any job2

E:\example>java switch3
Input any option(A/B/C)
C
Do any job3
```

Discussion: this example demonstrates option selections, if multiple options are displayed and we need to select any one, in that situation we have to type any one of options. It is an example of character data switching.

Different Loop statements:

Another name of loop is iteration, i.e., doing some jobs repeated times. There are three loop commands of loop and these are: for(…){} and while( ){} and do..while(). Unlike C/BASIC, Java does not support goto/go statement.

Concept of Loop: (Flowchart)

If we want to perform some jobs again and again, then loop commands are used. There are different types of loop commands as: while(), for() and do..while() while role of jobs are same performed by any of these loop statements, only the differene is that how to utilize the loop structure. The main point of learning loop is that starting number, ending number and condition when it will loop or not.

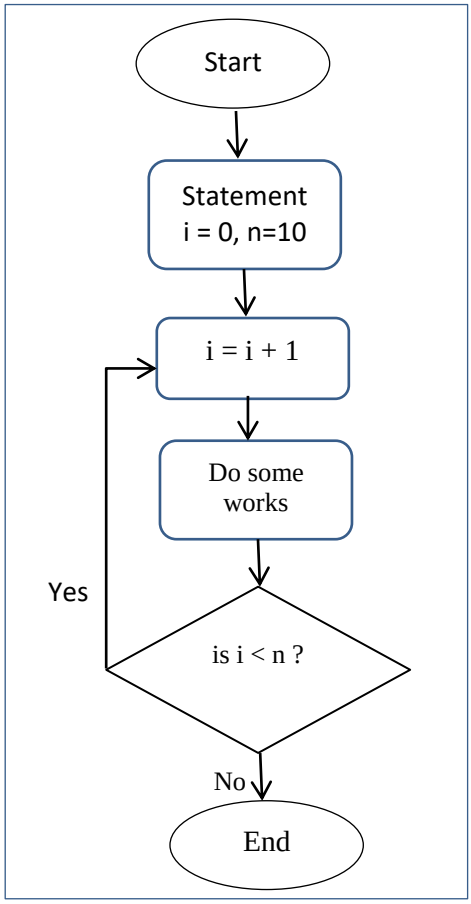

Fig. 3.2 a flowchart showing standard symbols of start-end, input-output, decision box, flow arrow of direction of movement of a program.

Fig. 3.2 . A flowchart shows start and end of a program. This flowchart uses a loop of ten times doing some works; if the loop counter 'i' reaches the value of 'n'(limit of loop) then it goes to End the program, else, it goes to the line of i = i + 1. It is the basic structure of a loop.

Problem: wap to input how many numbers to be printed and then using for loop print that many numbers from 1 to n. To input upper limit use Scanner class.

```
//ExFor1.java : to test loop statements using for(). To print count numbers from 0 to //p-1 where p is any integer number.
 import java.util.Scanner;
class ExFor1
{
        public static void main(String arr[])
        {
                int i;
                Scanner in=new Scanner(System.in);
                System.out.println("How many time? ");
                int p = in.nextInt();
                for (i=0;i<p;i=i+1 )
                {
                        System.out.print("\n Count is:" + i);
                }
        }
}
```

OUTPUT:

```
How many time?
10
 Count is:0
 Count is:1
 Count is:2
 Count is:3
 Count is:4
 Count is:5
 Count is:6
 Count is:7
 Count is:8
 Count is:9
```

Walkthrough the code: it is called analyzing the program line by line to check for any possible errors. In the above program, the critical section is the loop section to check whether it produces 9, 10 or 11 outputs. Let us walkthrough the code:

Step-1: i=0; p=10; 0<10;(logic test OK) so, enter into loop ;i=i+1=0+1=1;
Output: Count is:0 // note that 'i' is zero not one.

Step-2: i=1;p=10;1<10; (logic test OK) so, enter into loop ; i = i+1=1+1=2;
Output: Count is:1 // note that 'i' is one not two.

Step-3: i=2;p=10;2<10;(logic test OK) so, enter into loop;i=i+1=2+1=3;
Output: Count is: 2

Step-4: i=3;p=10;3<10; (logic test OK) so, enter into loop;i=i+1=3+1=4;
Output: Count is: 3

Step-5: i=4;p=10;4<10; (logic test OK) so, enter into loop;i=i+1=4+1=5;
Output: Count is: 4

Step-6: i=5;p=10;5<10; (logic test OK) so, enter into loop;i=i+1=5+1=6;
Output: Count is: 5

Step-7: i=6;p=10;6<10; (logic test OK) so, enter into loop;i=i+1=6+1=7;
Output: Count is: 6

Step-8: i=7;p=10;7<10; (logic test OK) so, enter into loop;i=i+1=7+1=8;
Output: Count is: 7

Step-9: i=8;p=10;8<10; (logic test OK) so, enter into loop;i=i+1=8+1=9;
Output: Count is: 8

Step-10: i=9;p=10;9<10; (logic test OK) so, enter into loop;i=i+1=9+1=10;
Output: Count is: 9

Step-11: i=10;p=10;10<10; (logic test FALSE) so, EXIT from the loop;

Note: if you observe carefully will see that inside the loop it prints the value of 'i' which is the first testing version of 'i', and that value is printed. When loop returns to beginning stage it increases the value of 'i' and then use this value for comparison to counter of loop. Else at the same time by name of i two values can't be stored. The mechanism of increment of i is the role of compiler as designed inside of it.

Problem: wap to input last number using the class *Scanner* and then print all odd numbers from 1 to last number using for loop.

```
//ExFor.java :to test loop statements using for(). To display odd numbers from 1 to p.
 import java.util.Scanner;
class ExFor
{
        public static void main(String arr[])
        {
                int i;
                Scanner in=new Scanner(System.in);
                System.out.println("How many time? ");
                int p = in.nextInt();
                for ( i=1; i<p; i = i+2 )
                {
                        System.out.print("\n Count is:" + i);
                }
        }
}
```

OUTPUT:

```
How many time?
10
Count is:1
 Count is:3
 Count is:5
 Count is:7
 Count is:9
```

Home work: * write the same program using while loop.
* Change the same program to print all even numbers.

Problem: wap to print all numbers from 0 to 10 using while loop.

```
//wh_loop.java : print all numbers from 1 to 10 using while( ) loop.
class wh_loop
{
        public static void main(String args[])
        {
                int n=10, i=0;
                //System.out.println("Sum of 1 to 10 = " + n);
```

```
                //start a loop to continue upto n times
                while(i<=n)
                {
                        System.out.print(  i +"," );
                        i++; //i=i+1
                }
        }
}
```

OUTPUT:
0,1,2,3,4,5,6,7,8,9,10,

Home work: * edit this program to print 1 to 10.
* write this program using for() loop.

Note: this program starts from zero, we assign this problem to students to edit and run this program that it will start from 1.

Problem: wap to print from 100 to 1 using while loop.

//wh_loop2.java : printing from 100 to 1 in a step down way

```
class wh_loop2
{
        public static void main(String args[])
        {
                int n=1, i=100;
                //start a loop to continue upto i times
                while(n <= i)
                {
                        System.out.print(  i +"," );
                        i--; //i=i-1; value of i is decreased by 1 in each steps.
                }
        }
}
```

OUTPUT:
100,99,98,97,96,95,94,93,92,91,90,89,88,87,86,85,84,83,82,81,80,79,78,77,76,75,74,73,72,71,70,69,68,67,66,65,64,63,62,61,60,59,58,57,56,55,54,53,52,51,50,49,48,47,46,45,44,43,42,41,40,39,38,37,36,35,34,33,32,31,30,29,28,27,26,25,24,23,22,21,20,19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,

Home work: * edit this program that it will display 10 numbers in each row.
* edit the same program using for() loop.
* draw a flow chart of this program.

Nested Loop:

If a loop is constructed within a loop, then it is called nested loop. To read or write data from/to a table having row and columns nested loop command is required. Say, there is a table like excel sheet,

1	2	3
1	2	3
1	2	3
1	2	3

We see there are four rows and three columns. To fill data in this table or Matrix, we need to construct an outer loop of 1-4 and inner loop 1-3 as:

```
for(int row = 1 ; row<=4 ; row++)
    for(int col=1 ; col<=3 ; col++)
        System.out.printf(col);
```

N.B. details of nested loop will be discussed in the section of array of next chapter.

Problem: wap to sum from 1 to 10 using while() loop.

```
//Sum1.java: sum from 1 to 10 using loop command while()
//this program sums 1 to 10 and computes summation within a
//variable 'sm', in each step summation result is stored within
//this variable and final result is stored within 'sm'.
class Sum1
{
    public static void main(String args[])
   {
          int n=10, i=0, sm=0;
          //System.out.println("Sum of 1 to 10 = " + n);
          //start a loop to continue upto n times
          while(i<=n)
          {
            sm= sm+i;
             //System.out.println("argument:" + i + "is:"+ st);
              i++; //i=i+1
          }
            System.out.println("Sum of 1 to 10 = " + sm);
    }
}
```

OUTPUT:

```
Sum of 1 to 10 = 55
```

Discussion: you can try Putting the println() line within the loop and check that summation results of ten steps will be printed. Note that while() is a looping command, if we want to do any job many times, then this loop command is used.
Similar to while(){} loop, another one is do..while() loop and its systax is as:

```
do{
  ......
  ......
}while(condition);
```

The execution rule is that first of all it will enter within the structure of the loop without checking the condition, so, once all statements inside will be executed, after that it will check the condition of looping and either will go to the starting point or come out from the loop.

Problem: wap to input a number from the keyboard and then check for whether the number is a prime or not; you should use for() loop.

```
//Prime.java:to test for a number is prime or not.
 import java.util.Scanner;
class Prime
{
   public static void main(String arr[])
   {
     int c;
     Scanner in = new Scanner(System.in);
    System.out.println("Input any number to be tested for prime ");
    int p = in.nextInt();
    for ( c = 2 ; c <= p - 1 ; c++ )
    {
       if ( p%c == 0 )
       {
          System.out.println(p+" : it is not prime number");
          break;
       }
    }
    if ( c == p )
          System.out.println(p+ " :it is prime number.");
  }
}
```

Discussion: there are many keyboard input methods under *Scanner* class like: nextInt(), nextLine(), nextByte(), nextShort(), next(), nextDouble(), nextFloat(), nextBoolean() etc.
If careful observation is done on this statement,
Scanner in = new Scanner(System.in);
We shall see at left side this ‘ *in* ’ is an object created from the class *Scanner*. Then, the constructor of the scanner “*Scanner()*” is initialized with a parameter “*System.in*”.

Home work/Assignments:

1. Write a program to sum odd numbers from 1 to 10:
It will sum: 1 + 3 + 5 + 7 + 9 =
Hints: increase the loop as i=i+2: where i starts from 1;
2. Write a program to sum even numbers from 1 to 10:
It will sum: 2 + 4 + 6 + 8 + 10 =
Hints: increase the loop as i = i + 2: where i starts from 0, zero.

Problem: wap to input/output data of three persons as name, age and salary using Scanner class and its methods. As loop command, do..while() should be used. It should be proved that Scanner class and nextLine() method has limitations to input multiple names. Then suggest the solution of this problem.

```
//InputScan1.java: Keyboard input using Scanner class and make a loop using
do..while() to input three students data.
import java.util.*;
//import java.io.DataInputStream;
public class InputScan1{
    public static void main(String args[]){
          int count=0;
           Scanner inp = new Scanner(System.in);
     do{
          System.out.println("Input Your Data:");
           System.out.print("Enter your name: ");
          String name = inp.nextLine();
          System.out.print("Enter your age: ");
          int age = inp.nextInt();
          System.out.print("Enter your salary: ");
          double sal = inp.nextDouble();
           System.out.println("Name: " + name);
           System.out.println("Your Age: " + age);
          System.out.println("Your Salary: " + sal);
           count++;
     }while(count<=2);
      inp.close();
  }
}
```

OUTPUT:

```
Input Your Data:
Enter your name: A R Rahman
Enter your age: 45
Enter your salary: 45678.60
Name: A R Rahman
Your Age: 45
Your Salary: 45678.6
Input Your Data:
Enter your name: Enter your age: 32   *error:no name
Enter your salary: 40000
Name:
Your Age: 32
Your Salary: 40000.0
Input Your Data:
Enter your name: Enter your age: 22    *error:no name
Enter your salary: 50000
Name:
Your Age: 22
Your Salary: 50000.0
```

Discussions: in this problem for inputting name, the class *Scanner* and its method *nextLine()* are used, but, it is seen 2nd and 3rd time name column is not asking to input names, it is the limitations of the *Scanner* class which can input a word properly but not a sentence. To solve this problem, we used the class of *DataInputStream* available under '*io*' package.

There are many overloaded constructors of the class "Scanner" having different numbers of arguments and of different data types as:
Scanner(InputStream source): has single parameter 'source' as type 'InputStream'.
Scanner(Readable source): single parameter 'source' but, as type of 'Readable'.
Scanner(InputStream source, String charsetName): two parameters are declared.
Scanner(String source): single parameter of String data type.
Scanner(File source): single parameter of type a data file 'File'.
Scanner(ReadableByteChannel source, String charsetName): two parameters are declared of different data types.
Scanner(Path source, String charsetName): two parameters

N.B. it is a good practice- to use '*close()*' after using any input/output stream.

Problem: wap to input name using the class *DataInputStream* , age and salary of a person using methods of *Scanner* class and then display. Make a loop using do..while() for inputting data of three persons.

```
//InputScan2.java: Keyboard input
import java.util.*;
import java.io.DataInputStream;
public class InputScan2{
    public static void main(String args[]){
    int count=0;
    //scanner for numbers input
    Scanner in = new Scanner(System.in);
    //for name input
     DataInputStream inp= new DataInputStream(System.in);
   try{
          do{
                System.out.println("Input Your Data:");
                System.out.print("Enter your name: ");
                String name = inp.readLine();
                System.out.print("Enter your age: ");
                int age = in.nextInt();
                System.out.print("Enter your salary: ");
                double sal = in.nextDouble();
                System.out.println("Name: " + name);
                System.out.println("Your Age: " + age);
                System.out.println("Your Salary: " + sal);
                count++;
          }while(count<=2);
          inp.close();in.close();
    }catch(Exception ee){}
```

```
  }
}
```

OUTPUT:

```
Input Your Data:
Enter your name: Ms.Marta Zurek
Enter your age: 23
Enter your salary: 200050.75
Name: Ms.Marta Zurek
Your Age: 23
Your Salary: 200050.75
Input Your Data:
Enter your name: Olga Jaksic
Enter your age: 33
Enter your salary: 300005.70
Name: Olga Jaksic
Your Age: 33
Your Salary: 300005.7
Input Your Data:
Enter your name: Alper Kerem
Enter your age: 35
Enter your salary: 250000.25
Name: Alper Kerem
Your Age: 35
Your Salary: 250000.25
```

Discussions: for general jobs of string manipulation like- input, output, convert to upper/lower, replacement etc., the class 'String' can be used. But, if multiple times string replacements are required in place of previous data, 'String' is not safe for those jobs. More efficient classes under language package are available and these are: *StringBuilder* or *StringBuffer*.

The class *scanner* is a member class of the package *util*. To input string with white spaces, Scanner is not sagfe as shown in previous of above example. There DataInputStream class is used with method readLine() to input full string. There are many other ways to input from keyboard using classes like- Scanner, BufferedReader, InputStreamReader, Console etc.: as mentioned below:

```
InputStreamReader ob = new InputStreamReader(System.in);
String name=ob.readLine();
Or
BufferedReader obj=new BufferedReader(new InputStreamReader(System.in));
Or
Console cb=System.console();
String nm=cb.readLine();
```

Table 3.2 Some Important JAVA packages:

Package Names	Role of the class
java.io	Class repository of all input and output activities.
java.awt	Contains all classes required for platform independent graphic jobs. Graphics features like: button, check box, pointer, mouse events etc.
java.lang	Most of the Java classes are under this package.
Java.util	Utility classes to handle events of tables and vectors
Java.applet	All classes required to run applets within any Browsers, eg. Chrome, Firefox, Edge, explorer, Netscape etc.
Java.net	All classes required for Network programming, connectivity etc.

Formatted Output:

Sometime, it is required to display numbers and texts in different formats. As output, any program generates some information and for requirements of different applications, information should be displayed as required formats. There are mainly four number systems used in programming languages:

Decimal: this system uses 0-9, ten digits to represent all numbers as:
0 1 2 3 4 5 6 7 8 9
Octal: this system uses 0-7, eight digits, to represent all numbers as: 0 1 2 3 4 5 6 7
Hexadecimal: This system uses, 0-9 ABCDEF, 16 digits and alphabets, to represent all numbers, these are: 0 1 2 3 4 5 6 7 8 9 A B C D E F
Binary: this system is mostly used for bitwise operations. Only two digits 0 and 1 are used in binary number system. Digital electronic circuit works on this binary number systems. For low level programming, binary number system is used.
In practical life, formatted output of any number or text is mostly used and a number of one format can be displayed in other number formats using % symbol. As shown below, the command will generate output of left side one digit and right side two digits of the decimal point. The last ‘f ’ denotes float type of data.

```
System.out.printf( "%1.2f", variable_name );
System.out.printf( "%1.2f", PI ); // output will be 3.14
```

Some examples of format specifiers are: %s, %f, %d, %12d, %8s, %1.2f, %13.6e etc.
Here, %d indicates an integer of decimal number system having base 10, %s is used for unformatted output, may be string or numbers, but it is displayed as type of string. An example of *format()* method is shown below:

```
String nm = "Ronit Das"
String str1= String.format("Student Name:  %s", nm);
%x : output is hexadecimal number
%f : output is float value
%d : output is integer value
```

String.format("|%8d|", n); //output will be displayed within a distance of 8 digits padded as right aligned.
String.format("|%-8d|", n);// output will be displayed within a distance of 8digits padded as left aligned.
System.out.println(String.format("%.3g", 532.972300)) ; // output is 532.9
System.out.println(String.format("%.4g", 532.972300)) ; // output is 533.0
%g is to use either %f or %e, whichever is shorter.

There are two string format methods in Java and they are as:
public static String format(String format, Object... args)
and
public static String format(Locale locale, String format, Object... args)

Problem: wap to demonstrate different features of formatted outputs.

```
//NumFormat.java:
//import java.lang.Math;
class NumFormat
{
	public static void main(String args[])
	{
		int x=572;
		String st="Example of Formatted Output:";
		System.out.println(String.format("Program of: %s\n", st));
		System.out.println(String.format("%.4g", 532.972300));
		System.out.println(String.format("|%12d|", x));
		System.out.println(String.format("|%-12d|",x));
	}
}
```

OUTPUT:

```
Program of: Example of Formatted Output:

533.0
|         572|
|572        |
```

Discussions: formatted output is required to display any text or numbers in different formats like limits of decimal digits, positioning of digits or texts, alignment etc. Here, first the texts are displayed from the position where %s is positioned; decimal number is rounded to four digits and adding one it has become 533.0, then, 572 is displayed as right aligned within a space area of 12; finally, in the last line 572 is displayed as left aligned within the space area of 12 as mentioned within the program.

Uppercase/Lowercase of texts:

```
String str = " My College name is abc"
System.out.println(str.toUpperCase());
System.out.println(str.toLowerCase());
```

Problem: wap to input name and age from keyboard using DataInputStream class and then print output.

```
//dataUpLow.java: this program shows how to convert to uppercase .
import java.io.DataInputStream;
class dataUpLow
{
        public static void main(String args[])
        {
                DataInputStream in= new DataInputStream(System.in);
                int x=0;
                String nm;
                try
                {
                        System.out.print("Enter your Name:  ");
                        nm=in.readLine();
                        System.out.print("Input your age: ");
                        x=Integer.parseInt(in.readLine());
                        System.out.println("Name is: " + nm.toUpperCase());
                        System.out.println("Age is: " + x);
                }
                catch(Exception ee){}
        }
}
```

OUTPUT:

```
Enter your Name:  Jim Marseng
Input your age: 36
Name is: JIM MARSENG
Age is: 36
```

Discussions: if it is required to make fonts as Bold/italic, that will be done through Font class of Graphics environment. Later it will be discussed.

There are many methods of DataInputStream for reading different types of data from input device. When we input through objects of DataInputStream, it is inputted as string, so , we need to convert those to corresponding number values as:

```
Float y=Float.valueOf(in.readLine()) . floatValue();
```

java.io.DataInputStream class has many methods to read data from the input device.

Table 3.3 methods of DataInputStream to read from keyboard.

Data Type	Method
boolean	readBoolean()
byte	readByte()
char	readChar()
double	readDouble()
float	readFloat()
void	readFully(byte[] b, int off, int len)
int	readInt()
String	readLine()

	Deprecated.
long	readLong()
short	readShort()

Regarding DataInputStream, details are available in documentation section of JAVA/Oracle:
https://docs.oracle.com/javase/7/docs/api/java/io/DataInputStream.html

Problem: wap to demonstrate different formats of date using different constructors of ***Date*** class.

```
//DemoDates.java: this program demonstrate printing dates in different formats.
import java.util.Date;
 class DemoDates
{
  public static void main (String args[])
  {
     Date date1, date2, date3;
     date1 = new Date(); // today's date /time
     System.out.println("First date: " + date1);
     System.out.printf("%s %tB %<te, %<tY", "Another Format:", date1);
           //year=1948, month=Apr=4th month, 2=2nd day of next month
            //7=hrs.,          55=mins., day is calculated by Gregorian Calendar.
     date2 = new Date(48, 4, 2, 7, 55);
     System.out.println("\nSecond date: " + date2);
           //mention format
     date3 = new Date("August 26 1960 10:30 PM");
     System.out.println("Third date: " + date3);
   }
}
```

OUTPUT:

```
First date: Thu Sep 02 13:22:23 IST 2021
Another Format: September 2, 2021
Second date: Sun May 02 07:55:00 IST 1948
Third date: Fri Aug 26 22:30:00 IST 1960
```

Discussion: JAVA starts counting of date time from the 1st January of 1900 year. The class Date has many Constructors with different numbers of arguments of different data types as shown below:

Date( ): it returns today's date and time when it is executed. Remember that it is not method but constructor. It allocates a **Date** object and initializes and represents the time to the nearest millisecond.
Date(int year, int month, int date): year is replaced by (year+1900) as per Gregoriam Calendar.
Date(int year, int month, int date, int hrs, int min): it has five integer arguments as shown in above example of date2 type of variable.
Date(int year, int month, int date, int hrs, int min, int sec): it has six arguments.

Date(long date): here date is in seconds which is computed from 1st January 00:00:00 IST and can find the date as per Gregorian Calender.
Date(String s): dates can be given as argument as String data type as "Mon, Mar 31, 1947" or as "Fri Aug 26 22:30:00 IST 1960" predefining the format of date as per requirements. Two examples of String type of arguments are shown below.
SimpleDateFormat dateformat = new SimpleDateFormat("day, mm dd yyyy");
String dateStr = new SimpleDateFormat("Mon, Mar 31 1947");

User defined Methods

A method is a structure and is defined to do some particular jobs. It can contain variable, method, constant etc. It can be declared within the main() method or outside the main() method also. There are two methods created- testPrint() and testSecond() outside the main() and these methods are known as user defined methods. User can create so many numbers of methods as required.
Another issue, variable overriding, is also demonstrated in this example. Careful observation will see that an integer type of variable 'test' is declared within main() method and again declared as integer in both the methods of testPrint() and testSecond(). When a variable is declared within a method it is private to that method and not accessible to other methods. Within the own method, when a version of test is printed, it uses the variable of its own and prints the value of that. It can be proved by the result of output as shown below.

```
//VarOverride.java: example of variable overriding.
//demonstration of creating user defined methods.
public class VarOverride
{
   public static void main(String[] args)
   {
     int test = 2019;
     System.out.println("Value of variable test is: " + test);
     testPrint(); // output  2020
     testSecond(test);
     System.out.println("test var of main() is: " + test);
   }
   public static void testPrint()
   {
     int test = 2020;//overriden variable
     System.out.println("In testPrint(), test is "+ test);
   }
   public static void testSecond(int test)
   {
    System.out.println(" test var value of main: " + test);
    test = 2021;//overridden variable
    System.out.println(" test var of testSecond() is: " + test);
   }
}
```

OUTPUT:

1. Value of test is: 2019
2. In testPrint(), test is 2020
3. test var value of main: 2019
4. test var of testSecond() is: 2021
5. Value of test of main() is: 2019

Discussion: we have put line numbers to outputs of the program to describe each lines easily. Fist line has printed 'test' value as 2019 which is the private variable of main() method. Then, testPrint() has called the defined method testPrint() and has entered within the structure of that where the 'test' variable is declared and assigned in this way:

```
int test = 2020;
```

So, the println() has printed the value of this version of 'test' as 2020; it is another example of variable *overriding.*
Then, the method testSecond(test) is called with an argument 'test'. Careful observation will see that this 'test' is the private variable in main() method having value test=2019. This value is passed to the structure of the testSecond(int test) method. In this method, another version of 'test' variable is declared and created as integer data type. What is the value of this version of 'test'? Answer is 2019.

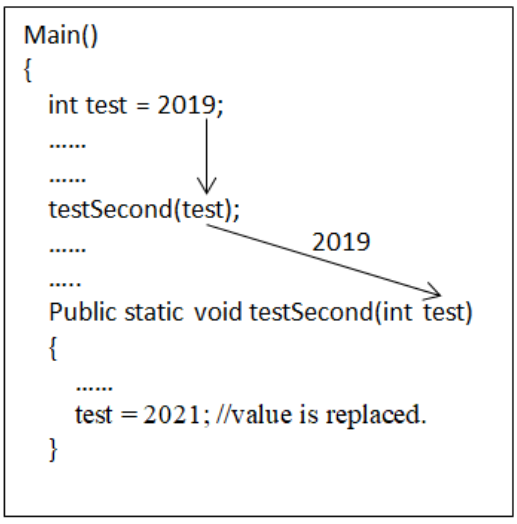

Fig. 3.3 a diagram of user defined method showing passing value through a variable. The same name 'test' of variable used in two places- within main() and testSecond() method which is called variable overridden.

More Explanations: the value test =2019 is passed to 'test' variable of ***testSecond()*** method, so, at this stage, test value is 2019 and that is printed by the first println() of the ***testSecond()*** method. After that, value of 'test' is replaced by 2021 and that is printed by the second ***println()*** statement. But, remember that replacement by **test=2021** is not considered as variable overridden. Variable overridden creates a new version of variable within that method; it is a fully new version of variable.

```
//power1.java: creating user defined method to create power of a number.
//This program also uses statements if..else..
//construction of a user defined function by name powerInt()

public class power1
{
   public static void main(String[] args)
  {
     int i;
     System.out.println("Computing x to the power of n" );
     for(i=0; i<10; i++)
    {
          System.out.printf("Power("+ i + " , " + i +") = %d%n ", powerInt(i, i));
    }
  }//end of main
  static int powerInt(int x, int n)
  {
     if(n == 0) return 1;
     int y = powerInt(x, n/2);
     if((n%2) != 0) return x*y*y;
     else return y*y;
   }
}//end of class
```

OUTPUT:

```
E:\example>javac power1.java

E:\example>java power1
Computing x to the power of n
Power(0 , 0) = 1
 Power(1 , 1) = 1
 Power(2 , 2) = 4
 Power(3 , 3) = 27
 Power(4 , 4) = 256
 Power(5 , 5) = 3125
 Power(6 , 6) = 46656
 Power(7 , 7) = 823543
 Power(8 , 8) = 16777216
 Power(9 , 9) = 387420489
```

Discussions: it is a program to generate output of n to the power of n, or n^n as shown in list of outputs. For understanding, careful observation is required; within main() module the user defined module ***powerInt(i, i)*** is called but the body of the created method is at end of the main(){} method. The for(..) loop generates increased value of 'I ' from 0 to 9 one after another. When i=0, and as ***powerInt(i, i)*** method has two parameters, now it is powerInt(0,0); two zeros are passed to two parameters of the method as x=0 and n=0 within the body construction section. Then next part is the role of computing the values. The result shows:

$0^0 = 1$; meaning that something power to zero is 1;
$1^1 = 1$;

Questions of chapter 3:

3.1 What do you mean by the identifier ‘static’ used to define main() method?

3.2 Explain ‘argument’ of a method. Are there any differences between main(String[] arg) and main(String arg[]) ?

3.3 How Boolean data type is used, give two examples.

3.4 Generally, for loop has three arguments like: for(int i=0; i<=9; i++), can we omit any argument and how?

3.5 Why unary operator can’t be used on two operators?

3.6 Mention a method name and a class name which can input a string type of data.

3.7 When is data type casting required?

Link of PPT of this book and all source codes:

`https://drive.google.com/drive/folders/1bMxMCaqPe0W35COAdn_-BrnV_uzQ-tNO?usp=sharing`

(Note: to get access, copy and paste this link to your browser. Both PPT and PDF version of presentation of this book are uploaded in Google Drive.)