CHAPTER 7 INPUT AND OUTPUT

Input represents data given to a program and output represents data displayed as a result of a program. We are already familiar with the following two statements to display the output:

```
System.out.print();
System.out.println();
```

Both of these statements are used to display the output on the screen. The difference between these two is that `print()` method keeps cursor in the same line after displaying the output and `println()` method throws cursor to the next line after displaying the output.

Input to a program can be given directly inside the program or from outside through the keyboard also. If the input is given in the program itself, every time the same input is used by the program and the same result can be expected. For example, in a program we write:

```
int a,b;
a = 10;
b = 15;
int c = a + b;
System.out.println(c);
```

In this example, the values of `a` and `b` are taken as `10` and `15`, respectively and the value of `c` is calculated and displayed. Even if this code is run several times, we get the value of `c` always as `25`. It means that this code is useful to perform addition of the two numbers: `10` and `15` only.

If we want to extend this code to find sum of any two given numbers, we should first accept those two numbers from the keyboard into the program. But how can we accept data from the keyboard and use it in a program? This question is answered in this chapter.

Accepting Input from the Keyboard

A stream is required to accept input from the keyboard. A stream represents flow of data from one place to another place. It is like a water-pipe where water flows. Like a water-pipe carries water from one place to another, a stream carries data from one place to another place. A stream can carry data from keyboard to memory or from memory to printer or from memory to a file. A stream is always required if we want to *move* data from one place to another.

Basically, there are two types of streams: input streams and output streams. Input streams are those streams which receive or read data coming from some other place. Output streams are those streams which send or write data to some other place.

All streams are represented by classes in `java.io` (input and output) package. This package contains a lot of classes, all of which can be classified into two basic categories: input streams and output streams.

Keyboard is represented by a field, called `in` in `System` class. When we write `System.in`, we are representing a standard input device, i.e. keyboard, by default. `System` class is found in `java.lang` (language) package and has three fields as shown below. All these fields represent some type of stream:

- **System.in:** This represents `InputStream` object, which by default represents standard input device, i.e. keyboard.
- **System.out:** This represents `PrintStream` object, which by default represents standard output device, i.e. monitor.
- **System.err:** This field also represents `PrintStream` object, which by default represents monitor.

Note that both `System.out` and `System.err` can be used to represent the monitor and hence any of these two can be used to send data to the monitor.

Important Interview Question

What is the difference between System.out and System.err ?

System.out and System.err both represent the monitor by default and hence can be used to send data or results to the monitor. But System.out is used to display normal messages and results whereas System.err is used to display error messages.

```
System.out.println("A normal message");
System.err.println("An error message");
```

So, the programmer can indicate what type of message he is displaying by using `System.out` oı `System.err`.

To accept data from the keyboard, i.e. `System.in`, we need to connect it to an input stream as somе input stream is needed to read data. Figure 7.1 shows the working in detail.

- Connect the keyboard to an input stream object. Here, we can use `InputStreamReader` tha can read data from the keyboard.

```
InputStreamReader obj = new InputStreamReader(System.in);
```

In this statement, we are creating `InputStreamReader` object and connecting the keyboar (`System.in`) to it.

- Connect `InputStreamReader` to `BufferedReader`, which is another input type of stream. W are using `BufferedReader` as it has got methods to read data properly, coming from th stream.

```
BufferedReader br = new BufferedReader(obj);
```

Here, we are creating `BufferedReader` object (`br`) and connecting the `InputStreamReader` obje (`obj`) to it.

In the above two steps, we got `BufferedReader` object (`br`). These two steps can be combined an rewritten in a single statement as:

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

- Now, we can read the data coming from the keyboard using `read()` and `readLine()` methods available in `BufferedReader` class.

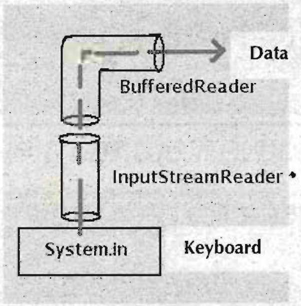

Figure 7.1 Reading data from keyboard

Accepting a Single Character from the Keyboard

Now, follow these steps in order to accept a single character from the keyboard.

- Create a `BufferedReader` class object (`br`).
- Then, read a single character from the keyboard using `read()` method as:

```
        char ch =  br.read();
```

Here, the `read()` method reads a single character from the keyboard but it returns its ASCII number, which is an integer. Since, this integer number cannot be stored into character type variable `ch`, we should convert it into `char` type by writing (`char`) before the method as:

```
        char ch = (char)br.read();
```

Here, `int` data type is converted into `char` type. Converting one data type into another data type is called *type casting* or simply *casting*. To convert `int` into `char`, we are writing (`char`) before the method or a variable. This is called *cast operator*.

If `read()` method could not accept a character due to some reason (like insufficient memory or illegal character), then it gives rise to a runtime error which is called by the name `IOException`, where `IO` stands for Input/Output) and `Exception` represents runtime error.

So, when `read()` method is giving `IOException`, as a programmer, it is our duty to do something in case of the exception. This is called *exception handling*. Since, at this moment we do not know how to handle exceptions, let us throw this exception without handling it by writing:

`throws IOException` at the side of the method where `read()` is used.

These concepts are used in the following program while receiving a single character from the keyboard.

Program 1: To accept and display a character from the keyboard

```
        //Accepting a single character from keyboard
        import java.io.*;
        class Accept
        {
                public static void main(String args[]) throws IOException
                {
                        //create BufferedReader object to accept data from keyboard
                        BufferedReader br = new BufferedReader(new
                        InputStreamReader(System.in));
                //ask for char and read it
                System.out.print("Enter a character: ");
                char ch = (char)br.read();

                //display the character
                System.out.println("You entered: "+ ch);
          }
     }
```

Output:

```
C:\> javac Accept.java
C:\> java Accept
Enter a character: A
You entered: A

C:\> java Accept
Enter a character: XXX
You entered: X
```

If you observe the output, you can understand that `read()` method is accepting only one character. It cannot be used to accept a string (a group of characters).

To read a string, we need `readLine()` method of `BufferedReader` class.

Accepting a String from Keyboard

Observe the statement, where `readLine()` method is called using `BufferedReader object (br)`:

```
String str = br.readLine();
```

Here, `readLine()` method accepts a string from the keyboard and returns the string into `str`. In this case, casting is not needed since `readLine()` is taking a string and returning the same data type. But this method can give rise to the runtime error: `IOException`, which can be thrown out without handling by using the statement:

```
throws IOException.
```

Program 2: Accepting a name (string) from the keyboard

```
//Accepting a string from keyboard
import java.io.*;
class Accept
{
   public static void main(String args[]) throws IOException
  {
    //create BufferedReader object to accept data from keyboard
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    //ask for string and read it
    System.out.print("Enter a name: ");
    String name = br.readLine();

    //display the string
    System.out.println("You entered: "+ name);
   }
}
```

Output:

```
C:\> javac Accept.java
C:\> java Accept
Enter a name: Vijay
You entered: Vijay
```

Accepting an Integer Value from the Keyboard

Now, let us follow these steps to accept an integer from the keyboard.

- First, we should accept the integer number from the keyboard as a string, using `readLine()` as:

```
String str = br.readLine();
```

- Now, the number is in `str`, i.e. in form of a string. This should be converted into an `int` by using `parseInt()` method of `Integer` class as:

```
int n = Integer.parseInt(str);
```

If needed, the above two statements can be combined and written as:

```
int n = Integer.parseInt(br.readLine());
```

Here, `parseInt()` is a static method in `Integer` class, so it cannot be called using class name as `Integer.parseInt()`

Let us observe a point here. We are not using casting to convert `String` type into `int` type. The reason is `String` is a class and `int` is a fundamental data type. Converting a class type into a fundamental data type is not possible by using *casting*. It is possible by using the method `Integer.parseInt()`. Remember, casting is useful to convert one fundamental data type into another fundamental data type or one class type into another class type. We cannot use casting to convert a class type into a fundamental data type or vice versa.

Program 3: Accepting an integer from keyboard.

```
//Accepting an int from keyboard
import java.io.*;
class Accept
{
   public static void main(String args[]) throws IOException
   {
     //create BufferedReader object to accept data from keyboard
     BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

     //ask for integer and read it
     System.out.print("Enter an int value: ");
     int num = Integer.parseInt(br.readLine());

     //display the int
     System.out.println("You entered: "+ num);
   }
}
```

Output:

```
C:\> javac Accept.java
C:\> java Accept
```

```
Enter an int value: 457890
You entered: 457890
```

Accepting a Float Value from Keyboard

Just like an integer value, we can also accept a float value in the following way:

```
float n = Float.parseFloat(br.readLine());
```

In this statement, we are accepting a float in the form of a string using `br.readLine()` and then passing the string to `Float.parseFloat()` to convert it into float. `parseFloat()` is a static method in `Float` class.

Program 4: Accepting a float number.

```
//Accepting a float value from keyboard
import java.io.*;
class Accept
{
  public static void main(String args[]) throws IOException
  {
    //create BufferedReader object to accept data from keyboard
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    //ask for float and read it
    System.out.print("Enter a float value: ");
    float n = Float.parseFloat(br.readLine());

    //display the float
    System.out.println("You entered: "+ n);
  }
}
```

Output:

```
C:\> javac Accept.java
C:\> java Accept
Enter a float value: 44.556
You entered: 44.556
```

Accepting a Double Value

We can accept a double value from the keyboard with the help of the following statement.

```
double n = Double.parseDouble(br.readLine());
```

Here, we are accepting a double in the form of a string using `br.readLine()` and then passing the string to `Double.parseDouble()` to convert it into primitive data type `double`. Let us know that `parseDouble()` is a static method in `Double` class.

Accepting Other Types of Values

Similarly, we can write different statements to accept many other data types from the keyboard as follows:

❑ To accept a byte value:

```
byte n = Byte.parseByte(br.readLine());
```

- To accept a short value:

```
short n = Short.parseShort(br.readLine());
```

- To accept a long value:

```
long n = Long.parseLong(br.readLine());
```

- To accept a boolean value:

```
boolean  x = Boolean.parseBoolean(br.readLine());
```

In the above discussion, we used the classes, such as Byte, Short, Integer, Long, Float, Double, and Boolean, which belong to java.lang package. These classes are also called *wrapper classes.* We will have a complete look at wrapper classes in later chapters.

Since we know how to accept various data types, let us write a small program to accept an employee details and display them again.

Program 5: Accepting and displaying employee details.

```
//Employee details
import java.io.*;
class EmpData
{
        public static void main(String args[]) throws IOException
        {
                //create BufferedReader object to accept data
                BufferedReader br = new BufferedReader(new
                InputStreamReader(System.in));

                //accept employee details
                System.out.print("Enter id: ");
                int id = Integer.parseInt(br.readLine());

                System.out.print("Enter sex (M/F): ");
                char sex = (char)br.read();

                System.out.print("Enter name: ");
                String name = br.readLine();

                //display the employee details
                System.out.println("Id= "+ id);
                System.out.println("Sex= "+ sex);
                System.out.println("Name= "+ name);
        }
}
```

Output:

```
C:\> javac EmpData.java
C:\> java EmpData
Enter id: 10
Enter sex (M/F): M
Enter name: Id= 10
Sex= M
Name=
```

Please observe the output of Program 5. After accepting sex of the employee, it is not accepting the name of employee and is displaying blank for name. The reason is that we used read() method to accept the sex value and then readLine() is used to accept the name. When we type M for sex and press Enter, then it releases a \n code. So at sex column, we are giving two characters M and \n.

But, `read()` method takes only the first character and rejects the next character, i.e. `\n`, which is trapped by the next `readLine()` method, as shown in Figure 7.2. So, name is not accepted by `readLine()` as it already contains `\n`.

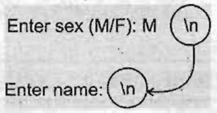

Figure 7.2 The readLine() method trapping the /n code

There are two ways to overcome this problem.

- ❑ One solution is to use `readLine()` method to accept the single character while accepting the sex of employee. We can write it as follows:

```
System.out.print("Enter sex (M/F): ");
char sex = br.readLine().charAt(0);
```

In the above statement, we are first accepting the input for sex as a string, i.e. `M\n` using `readLine()`. Then, we are retrieving the 0th character, i.e. only `M` from it and returning it into `sex` variable. This is done by `charAt(0)` method.

- ❑ Clearing the `\n` code from `BufferedReader` object, so that the character is not carried into next `readLine()`.

For this purpose, we can use `skip()` method of `BufferedReader`, which helps in skipping a specified number of characters. Suppose we take `\n` as two characters; now to skip them, we can write `br.skip(2);`

Now let us see the modified version of the program.

Program 6: Accepting and displaying employee details – version 2

```
//Employee details
import java.io.*;
class EmpData
{
	public static void main(String args[]) throws IOException
	{
		//create BufferedReader object to accept data
		BufferedReader br = new BufferedReader(new
		InputStreamReader(System.in));

		//accept employee details
		System.out.print("Enter id: ");
		int id = Integer.parseInt(br.readLine());

		System.out.print("Enter sex (M/F): ");
		char sex = (char)br.read();   //Solution1: char sex =
		br.readLine().charAt(0);

		//Solution2: skip 2 characters
		br.skip(2);

		System.out.print("Enter name: ");
		String name = br.readLine();

		//display the employee details
		System.out.println("Id= "+ id);
		System.out.println("Sex= "+ sex);
		System.out.println("Name= "+ name);
	}
}
```

Output:

```
C:\> javac EmpData.java
C:\> java EmpData
Enter id: 10
Enter sex (M/F): M
Enter name: Kumar
Id= 10
Sex= M
Name= Kumar
```

From the above program, we can understand that in Java if we want to accept input, we can accept only one input at a time. For example, we are accepting id first, then sex, and then name of the employee—one by one. There is no way of accepting all the three types at a time (in a line) from the keyboard in Java. This is possible in C language using `scanf()` and in C++ using `cin` functions.

Accepting Different Types of Inputs in a Line

It is possible to simulate `scanf()` of C language in Java, where we can receive several inputs at a time in a single line. To achieve the effect of `scanf()`, we can use `StringTokenizer` class of `java.util` (utility) package. `StringTokenizer` is useful to break a string into small pieces called *tokens*. Let us follow these steps to understand how it is done.

- First receive a string `str` from the keyboard, which contains different types of inputs. Assume that the inputs are separated by commas.

```
String str = br.readLine();
```

- Pass this string `str` to `StringTokenizer` object, so that it will be broken into pieces wherever a comma is found. These tokens will be stored in `StringTokenizer object st`.

```
StringTokenizer st = new StringTokenizer(str, ",");
```

In the above statement, `str` is the string which is split into tokens and `which` represents the character from where to split the string. This comma is also called *delimiter*. Suppose we want to split the string where a space is found, we can use a space as delimiter as:

```
StringTokenizer st = new StringTokenizer(str, " ");
```

- Collect the individual tokens from `st` using `nextToken()` method of `StringTokenizer class`.

```
String token = st.nextToken();
```

- These individual tokens represent the different types of inputs given. These tokens can be converted into corresponding data types and can be used in the program.

By using these steps, let us now write a Java program to accept name, age, and salary of a person at a time and display them again.

Program 7: To accept different types of input in a line at a time from the keyboard, just like one can do using `scanf()` in C

We assume that the different values of input are separated by commas.

```
//Accepting different inputs in a line
import java.io.*;
import java.util.*;
class Different
```

```
{
        public static void main(String args[]) throws IOException
        {
                //to accept data from keyboard
                BufferedReader br = new BufferedReader(new
                InputStreamReader(System.in));

                //ask for input - separated by commas
                System.out.print("Enter name, age, salary: ");

                //accept input into a string
                String str = br.readLine();

                //use StringTokenizer to split input at commas
                StringTokenizer st = new StringTokenizer(str, ",");

                //we will have 3 tokens as strings
                //first token represents name, second one age, third one salary
                String s1 = st.nextToken();
                String s2 = st.nextToken();
                String s3 = st.nextToken();

                //trim any spaces before and after the tokens
                s1 = s1.trim();
                s2 = s2.trim();
                s3 = s3.trim();

                //convert s1 into string, s2 into an int and s3 into a float
                String name = s1;
                int age =  Integer.parseInt(s2);
                float sal = Float.parseFloat(s3);

                //display the entered data
                System.out.println("Name= "+ name);
                System.out.println("Age= "+ age);
                System.out.println("Salary= "+ sal);
        }
}
```

Output:

```
C:\> javac Different.java
C:\> java Different
Enter name, age, salary: Vijaya Gopal, 25, 12000.75
Name= Vijaya Gopal
Age= 25
Salary= 12000.75
```

Now let us write a program to accept any two numbers from keyboard and find the results o addition, subtraction, multiplication, and division operations on them.

Program 8: To perform different arithmetic operations on given numbers.

```
//Performing arithmetic operations.
import java.io.*;
import java.util.*;
class Arithmetic
{
        public static void main(String args[]) throws IOException
        {
                //to accept data from keyboard
                BufferedReader br = new BufferedReader(new
                InputStreamReader(System.in));

                //take input into str from keyboard
                System.out.print("Enter two numbers: ");
                String str = br.readLine();

                //split the string at comma
                StringTokenizer st = new StringTokenizer(str, ",");

                //take the two tokens into s1, s2
                String s1 = st.nextToken();
                String s2 = st.nextToken();

                //trim the spaces in s1, s2
                s1 = s1.trim();
                s2 = s2.trim();

                //convert s1 and s2 into double type and store in n1, n2
                double n1 = Double.parseDouble(s1);
                double n2 = Double.parseDouble(s2);

                //perform the arithmetic operations
                System.out.println("Result of addition: "+ (n1 + n2));
                System.out.println("Result of subtraction: "+ (n1 - n2));
                System.out.println("Result of multiplication: "+ (n1 * n2));
                System.out.println("Result of division: "+ (n1 / n2));
        }
}
```

Output:

```
C:\> javac Arithmetic.java
C:\> java Arithmetic
Enter two numbers: 20.5, 10
Result of addition:  30.5
Result of subtraction: 10.5
Result of multiplication: 205.0
Result of division: 2.05
```

Let us write another program to test whether a given year is leap or not. The following logic can be used for doing so:

- If the year is representing the beginning of a new century like 1900, 2000, 2100, 2200, etc. it should be divisible by 400. Then it is called leap year. To know whether the given year is century year or not, we should divide it by 100. If it is divisible by 100, then it is a century year.
- If the year is not a century year like 1998, 2007, 2010, etc. it should be divisible by 4. Then it becomes leap year.
- If any of the above two cases are satisfied, then that year is called leap year, otherwise it is not a leap year.

From this discussion, we can conclude that a year is leap only when any one of the following two conditions is true.

- if (year % 100 ==0 && year % 400 == 0)
- if (year % 100 != 0 && year % 4 ==0)

Please observe how the same steps are implemented in the logic of Program 9.

Program 9: To accept a year number from the keyboard and test if it is leap or not.

```
//Leap year or not
import java.io.*;
class Leap
{
        public static void main(String args[]) throws IOException
        {
                //accept year
                BufferedReader br = new BufferedReader(new
                    InputStreamReader(System.in));

                    System.out.print("Enter year no: ");
                    int year = Integer.parseInt(br.readLine());

                    //if it is century year and divisible by 400
                    if(year % 100 ==0 && year % 400==0)
                    System.out.println("It is leap");
                    //if it is not a century year and divisible by 4
                    else if(year % 100 !=0 && year % 4==0)
                    System.out.println("It is leap");
                    //other wise, it is not leap year
                    else System.out.println("It is not leap");
            }
    }
```

Output:

```
C:\> javac Leap.java
C:\> java Leap
Enter year no: 2007
It is not leap
C:\> java Leap
Enter year no: 1900
It is not leap
C:\> java Leap
Enter year no: 2000
It is leap
```

Let us now write another program in which we will learn to create Fibonacci series. Fibonacci numbers are the numbers which follow the series:

```
0,1,1,2,3,5,8,13,…
```

The first two Fibonacci numbers are taken as 0 and 1. The next Fibonacci number is generated by adding these two. So we get 0+1 = 1. The next Fibonacci number also follows the same rule. Any Fibonacci number is the sum of its two previous Fibonacci numbers.

In order to generate this series, first we should take two Fibonacci numbers as `f1` and `f2`. The next Fibonacci number, say `f`, will be obtained by adding these two numbers.

```
f1 = 0;
f2 = 1;
f = f1+ f2;   //the next number is sum of the two previous numbers
```

Now, the two recent Fibonacci numbers are `f2` and `f`. So, to get the next Fibonacci number, we should add them by taking them as `f1` and `f2`:

```
f1 = f2;  //take f2 as f1
f2 = f;   //take f as f2
f= f1+ f2; //next Fibonacci no.
```

To repeatedly generate the Fibonacci numbers, we can use the above logic in a loop, as shown below:

```
while(count < n)  //here, count represents the number of fibonaccis generated
{
        f1 = f2;
        f2 = f;
        f = f1+f2;
        System.out.println(f);  //display the new Fibonacci no.
```

```
            count++;  //increment count since one more is generated
        }
```

The same logic is shown in Figure 7.3 and also in subsequent Program 10.

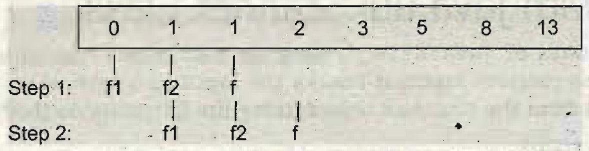

Figure 7.3 Logic to display Fibonacci numbers

Program 10: Generating Fibonacci numbers

```
//Fibonacci number series
import java.io.*;
class Fibo
{
     public static void main(String args[]) throws IOException
     {
            //accept how many fibonaccis needed
            BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));

            System.out.print("How many fibonaccis? ");
            int n = Integer.parseInt(br.readLine());

            //take first two fibonaccis as 0 and 1
            long f1 =0, f2 =1;

            System.out.println(f1);
            System.out.println(f2);

            //find next fibonacci no.
            long f = f1+f2;
            System.out.println(f);

            //Already 3 fibonaccis are displayed. So count will start at 3
            int count = 3;
            while(count < n)
            {
                   f1 = f2;
                   f2 = f;
                   f = f1+f2;
                   System.out.println(f);
                   count++;
            }
     }
}
```

Output:

```
C:\> javac Fibo.java
C:\> java Fibo
How many fibonaccis? 10
0
1
1
2
3
5
8
13
21
```

```
34
```

Reading Input with java.util.Scanner Class

We can use `Scanner` class of `java.util` package to read input from the keyboard or a text file. When the `Scanner` class receives input, it breaks the input into several pieces, called *tokens*. These tokens can be retrieved from the `Scanner` object using the following methods:

- `next()` – to read a string
- `nextByte()` – to read byte value
- `nextInt()` – to read an integer value
- `nextFloat()` – to read float value
- `nextLong()` – to read long value
- `nextDouble()` – to read double value

To read input from keyboard, we can use `Scanner` class as:

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

Now, if the user has given an integer value from the keyboard, it is stored into the `Scanner` objec (`sc`) as a token. To retrieve that token, we can use the method: `sc.nextInt()`. The followin program will make this concept clear.

Program 11: Let us write a program to read different types of data separated by space, from th keyboard using the `Scanner` class.

```
//Scanner to scan the input from keyboard
import java.util.Scanner;
class Ex3
{
	public static void main(String args[])
	{
		System.out.print("Enter id name sal: ");
		Scanner sc = new Scanner(System.in);

		int id = sc.nextInt();
		String name= sc.next();
		float sal = sc.nextFloat();

		System.out.println("Id= "+ id);
		System.out.println("Name= "+ name);
		System.out.println("Sal= "+ sal);
	}
}
```

Output:

```
C:\> javac Ex3.java
C:\> java Ex3
Enter id name sal:10 Gopal 8900.50
Id= 10
Name= Gopal
Sal= 8900.5
```

Displaying Output with System.out.printf()

To format and display the output, `printf()` method is available in `PrintStream` class. This method works similar to `printf()` function in C. We know that `System.out` returns `PrintStream` class object, so to call the `printf()` method, we can use: `System.out.printf()`.

The following format characters can be used in `printf()`:

- `%s` - string
- `%c` - char
- `%d` - decimal integer
- `%f` – float number
- `%o` – octal number
- `%b`, `%B` – boolean value
- `%x`, `%X` – hexadecimal number
- `%e`, `%E` – number in scientific notation
- `%n` – new line character

An example for using `printf()` is given below:

```
System.out.printf("Salary= %f", sal);
```

Here, the string `Salary=` will be displayed as it is in the output. After this, we have `%f`, which represents a format character to display a float value, i.e. `sal` value. So if `sal` variable has `8900.75`, then the output displayed by the above statement will be:

```
Salary= 8900.75
```

Program 12: To understand the use of `printf()`, let us write a Java program.

```
//printf() in Java
class Ex1
{
      public static void main(String args[])
      {
            String s1= "Hello";
            int n= 65;
            float f = 15.1234f;

            System.out.printf("String= %s%nnum= %d%nhexa decimal= %x%nfloat=
            %f", s1, n, n, f);

      }
}
```

Output:

```
C:\> javac Ex1.java
C:\> java Ex1
String= Hello
num=  65
hexa decimal= 41
float= 15.123400
```

Displaying Formatted Output with String.format()

If we want only a string that consists of formatted output, then we can take the help of `format()` method of `String` class. The format characters supported by `System.out.printf()` are also usable with `format()` method. Since, format() is a static method, we can call it as: `String.format()`. This method returns a string that contains the formatted output which can be processed and used as the programmer wants it.

Program 13: Understanding `format()` method to obtain a string that consists of formatted output.

```
//getting formatted output into a string.
class Ex2
{
        public static void main(String args[])
        {
                int i = 65;
                String s = "Hai";
                char ch = 'A';

                //format the output and get into str
                String str = String.format("i=%d%ns=%s%nch=%c", i,s,ch);
                System.out.println(str);
        }
}
```

Output:

```
C:\> javac Ex2.java
C:\> java Ex2
i=65
s=Hai
ch=A
```

Conclusion

We know already that `System.out.print()` and `System.out.println()` methods are useful to send output to the monitor. In many programs, the user wants to input values or data required by the program from the keyboard and hence `BufferedReader` class methods `read()` and `readLine()` are very useful. In this chapter, we discussed the techniques to accept input from the keyboard and wrote some programs to illustrate the same.