1. **System.out.println Method.**
2. **Reading Runtime values from KeyBoard.**

**System.out.print|ln|f Methods**

**1.System.out.println():-**

Where System is class name . It is declared final class.

Out is static and final variable of System and is type of PrintStream class.

**a.print():-** This is method of PrintStream class.

Syntax:-

Public void print(argument);

The print() method displays the result on the console and retains the cursor in the same line. **It works only with an argument.**

Example:-

class Clerk extends BankAccount

{

public static void main(String arg[])

{

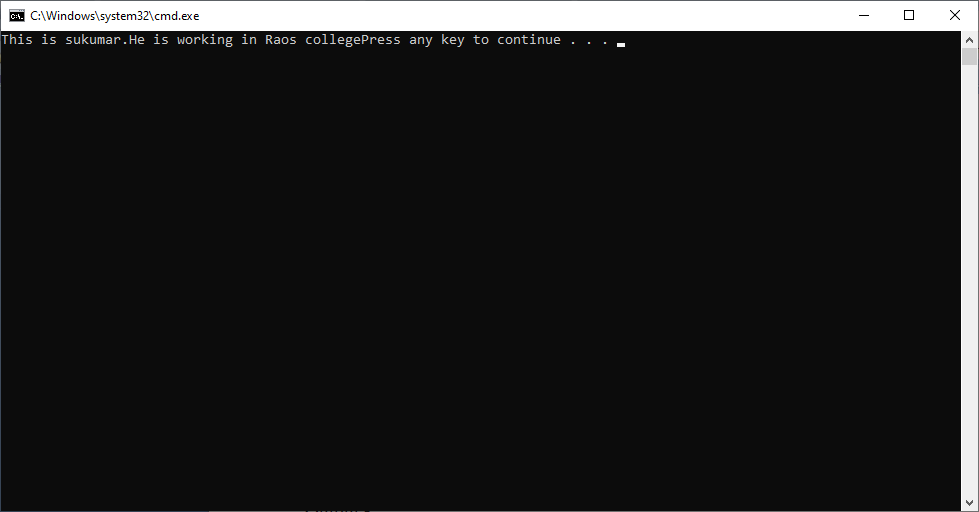
System.out.print("This is sukumar.");

System.out.print("He is working in Raos college");

}

}

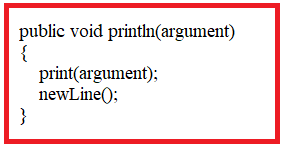
Output:-



**2.println():-**The println() method also displays the result on the console but moves the cursor to the next line. **It can also work without arguments**.

Syntax: public void println([Argument]);

Note:- The println() method does not has logic separately to print. Inside println() method, the print() method and new line() method is invoked.



Example:-

class Clerk extends BankAccount

{

public static void main(String arg[])

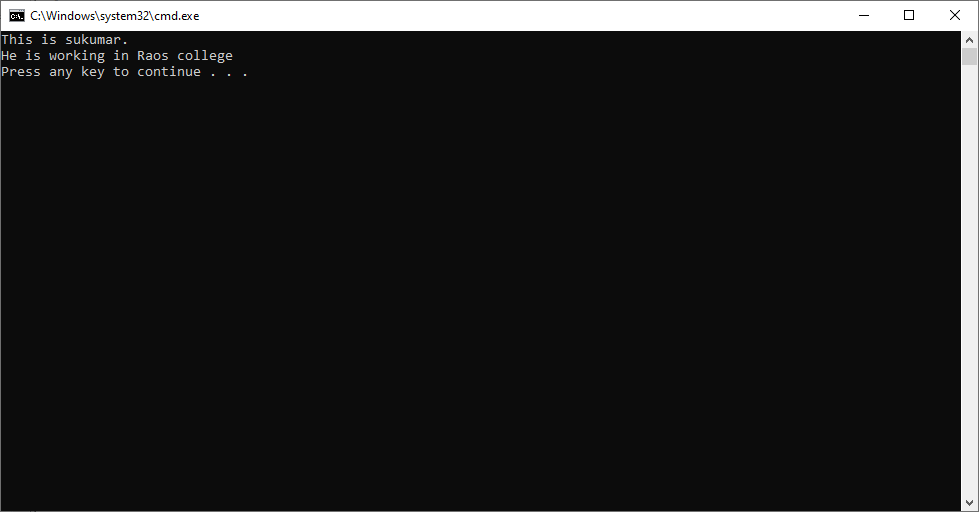
{

System.out.println("This is sukumar.");

System.out.println("He is working in Raos college");

}

}



Exercise: Develop a program to display following statements in 3 lines.

Sukumar

Atmakuru

Mca

Nellore

Rajaka Street.

Expected output:

Sukumar Atmakuru

Mca

Nellore , Rajaka street.

class Clerk extends BankAccount

{

public static void main(String arg[])

{

System.out.print("Sukumar");

System.out.println(" Atmakuru");

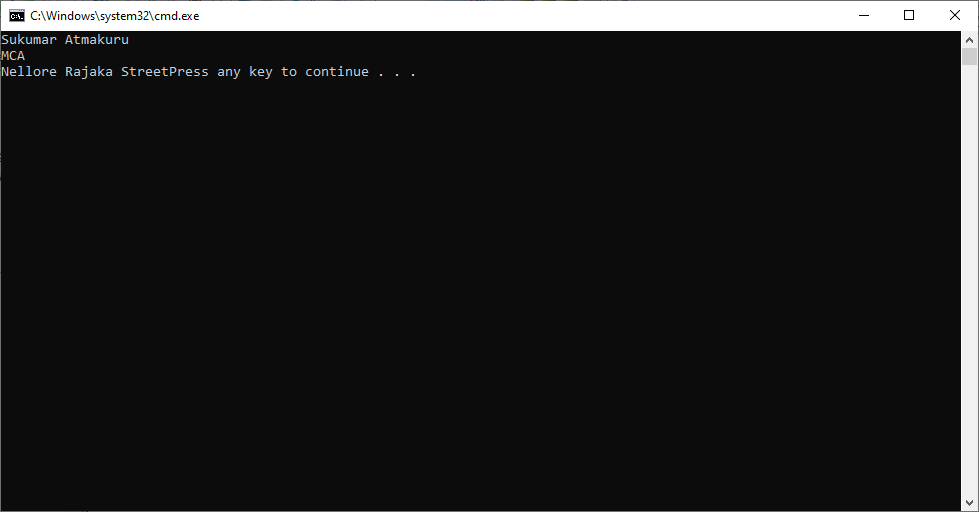
System.out.println("MCA");

System.out.print("Nellore");

System.out.print(" Rajaka Street");

}

}



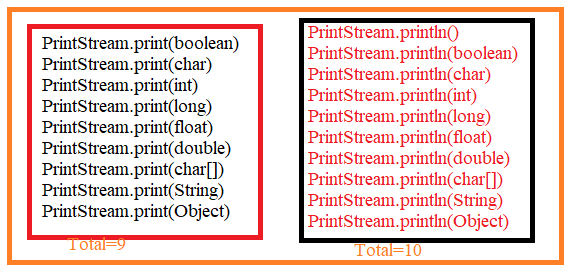
Note:- The

Print(“\n”);

Println();

Both are same.

The print and println methods are overloaded methods.



**3. Printf():-** It displays arguments on console. **It does not move cursor to next line.**

**Syntax:1** public void printf(String format\_specifier[s][, argument[s]]);

Syntax:2 public void printf(locale[, String format\_specifier[s], argument[s]]);

When u run printf() method with formating specifiers,inside printf() logic all formating specifiers are replaced with given values from left to right. The extra values are ignored.

Example:1

class Clerk extends BankAccount

{

public static void main(String arg[])

{

System.out.printf("sukumar");

System.out.printf("%d ",20);

System.out.printf("%f ",34.5);

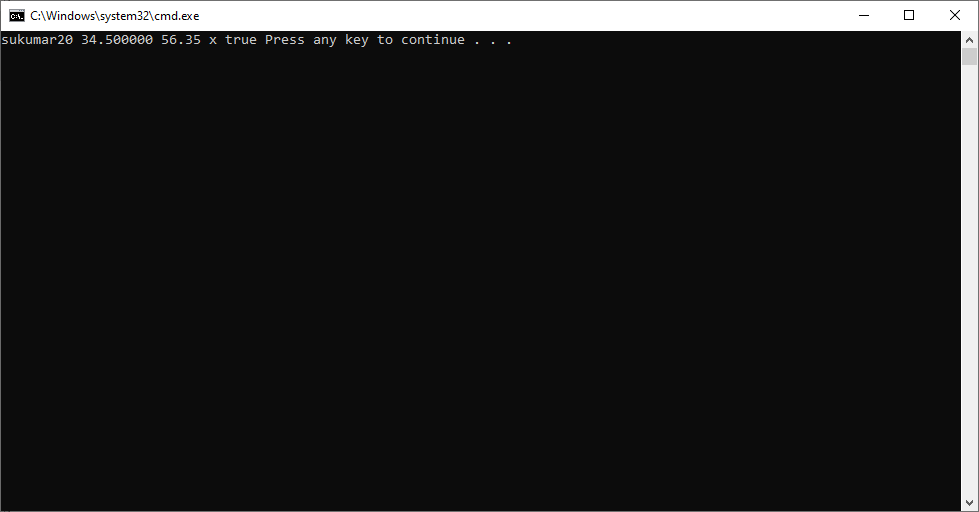
System.out.printf("%.2f ",56.3456);

System.out.printf("%c ",'x');

System.out.printf("%b ",true);

}

}



Example:2

class Clerk extends BankAccount

{

public static void main(String arg[])

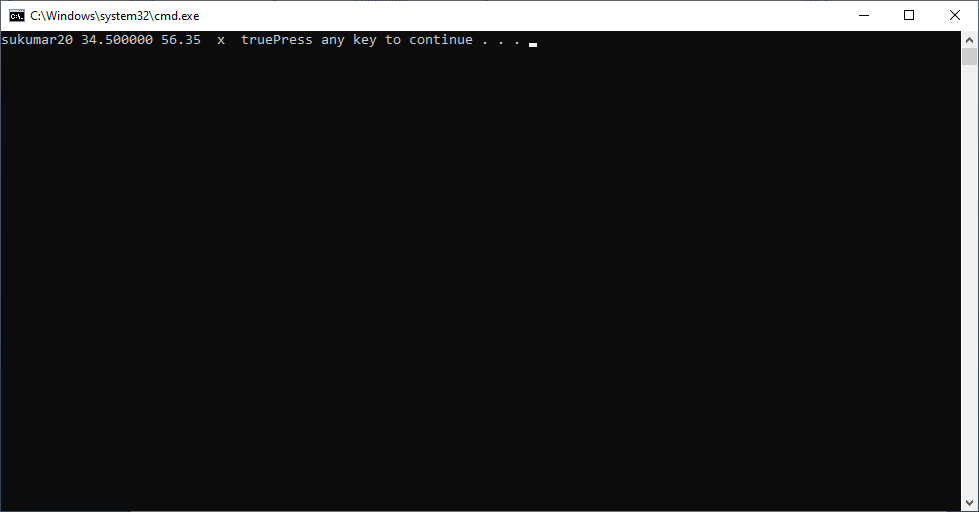
{

System.out.printf("sukumar");

System.out.printf("%d %f %.2f %c %b",20,34.5f,56.3456f,'x',true);

}

}



Test cases:

1. System.out.printf(“%d”,10); output:10
2. System.out.printf(“Hi %d”,10); output: Hi 10
3. System.out.printf(“Hi %d %d”,20); // It leads to RE. because no.of values are less than no.of format specifiers.
4. System.out.printf(“%d”,20,30,40); output:10 Extra values are ignored by printf().
5. System.out.printf(“%d”); // It leads to RE. Because there is no value to corresponding format specifier.

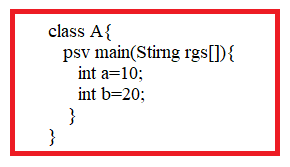
**READING THE RUNTIME VALUES FROM KEYBOARD**

**1.**Hardcoding means Assigning Values directly to variables in program.

**2.HardCoded Value:-** The value which is directly stored in variable for performing calculations is technically called hard code value.

**3.HardCoded Application:** The application in which values are directly specified to perform validation and calculation is technically called hardcoded application.

Example:



**4.Runtime Value:-** The value which is passed to program at run time is called runtime value.

**5.Dynamic Application:**The application which reads input from keyboard at the time of execution is called “**dynamic application”**.

**6.Problems with HardCoded application:**

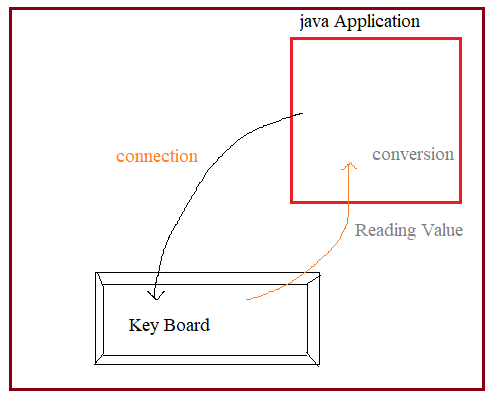
**a.** The application can always be executed with same values specified in program.

b. if want to change a values, we must modify values in source code then we must recompile the source code.

Real Time Example:

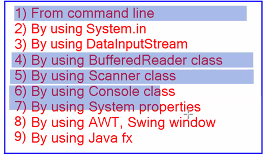
Let Us assume, ATM application is static Application. It does ‘t takes with draw amount from User at run time. Problem is that Bank customer always draw same amount from his account. He can’t take less amount and he can’t take greather amount than hard code amount which is specified in ATM application.

**7. Different Activities we must do read values from keyboard:**

****

**Step1:** Java program has to be connected with keyboard.For Connecting , we

Must choose one of nine approaches. We must choose approach as per ‘project requirement’.



**Step2:** Java program has to read value from keyboard. For reading value, In each approach there is one predefined method.

**Step3:**Taken value from keyboard is always String DT. Convert value into its original DT.

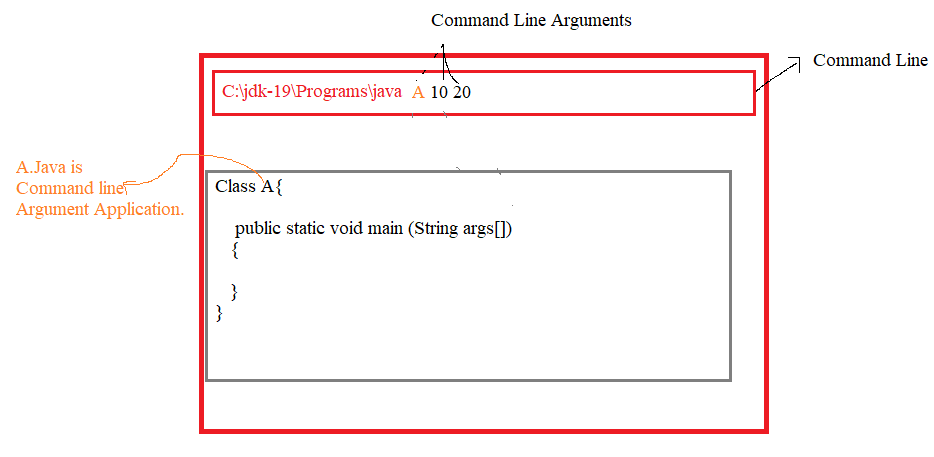
For converting , We must user wrapper classes and their methods.

**8. From Command Line:**

8.1.CommandLine:- In console window / In command prompt, we run commands javac and java for compiling and executing java program. The line from which we type commands is called command line.

8.2.CommandLine Arguments:- The values them we pass to program from command line is said to be commandline arguments.

8.3.CommandLine Argument Application:- The application which reads runtime values from command line to execute its logic is called commandline argument Application.



8.4. How Command line Arguments are passed into Application?

When you run a class(A) with command line Arguments(10,20),

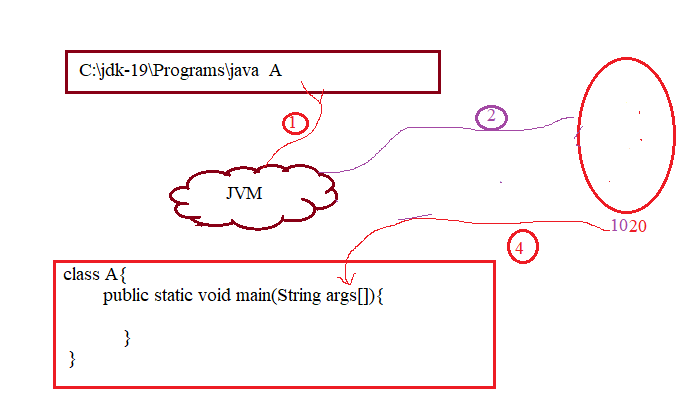
1.JVM reads the arguments.

2. JVM creates String type array.

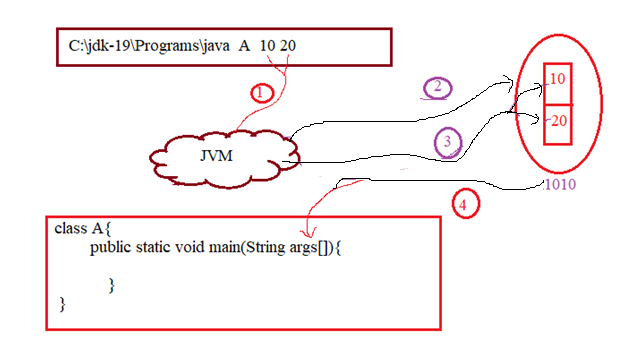
3. If there are arguments, JVM stores commandline arguments as **“string values”** in Array. Otherwise Array is empty array.

4. JVM invokes main() method. JVM passes Array object Reference as argument to main(). This array object reference will be stored in ‘args’.

Case:1



Case:2



Example:1

class A

{

public static void main(String[] args)

{ int i;

for(i=0;i<args.length;i++)

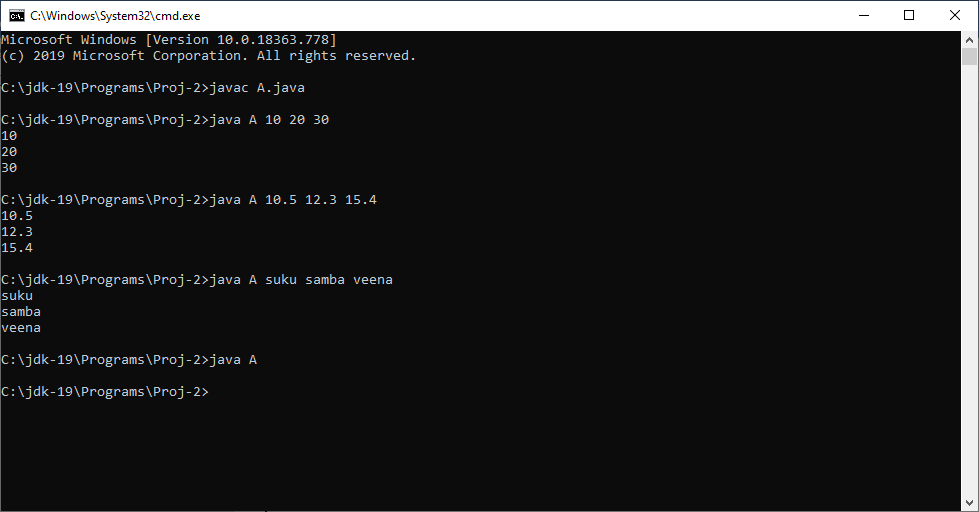
{

System.out.println(args[i]);

}

}

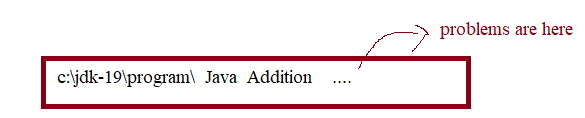
}



8.5.Problems with command line arguments:

a. The End user does not know that he has to pass values to program.

b. The End user does not know that how many values he has to pass & what type values he has to give.



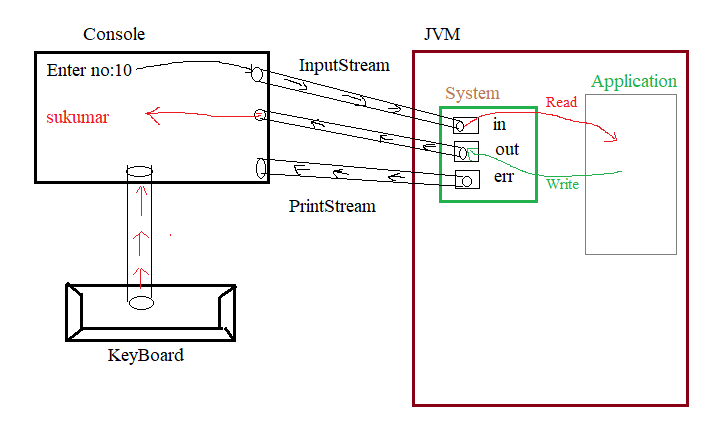
To avoid this, we must go to Second Approach(System.in).

**9.System.in:-** The system is a predefined class available in ‘**java.lang’** package. The system class has following 3 static variables.

1. static InputStream in: It contains the reference of InputStream object.

2. static PrintStream out: It contains the reference of PrintStream object.

3. static PrintStream err: It contains the reference of PrintStream object.

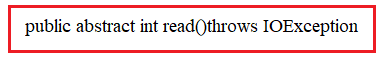


The JVM creates InputStream , PrintStream objects and JVM attaches them with console. During loading of System class, JVM put references of those objects in specific variables of System class.

Java application uses an print stream to write data to a destination; it may be a file, an array, peripheral device(console) or socket.

Java application uses an Input stream to read data to a destination; it may be a file, an array, peripheral device(console) or socket. The java application uses read() method of inputStream object.

Syntax:



It reads the one byte from inputStream(destination). It returns ascii value of that byte.

Example:1

import java.io.\*;

class sample

{

public static void main(String[] args) throws IOException

{

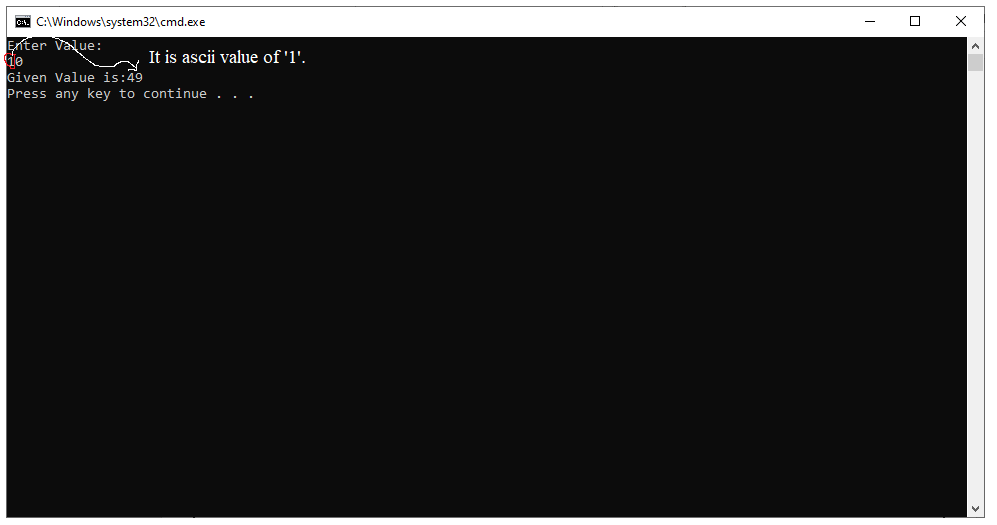
System.out.println("Enter Value:");

int a=System.in.read();

System.out.println("Given Value is:"+ a);

}

}



Example:2

import java.io.\*;

class sample

{

public static void main(String[] args) throws IOException

{

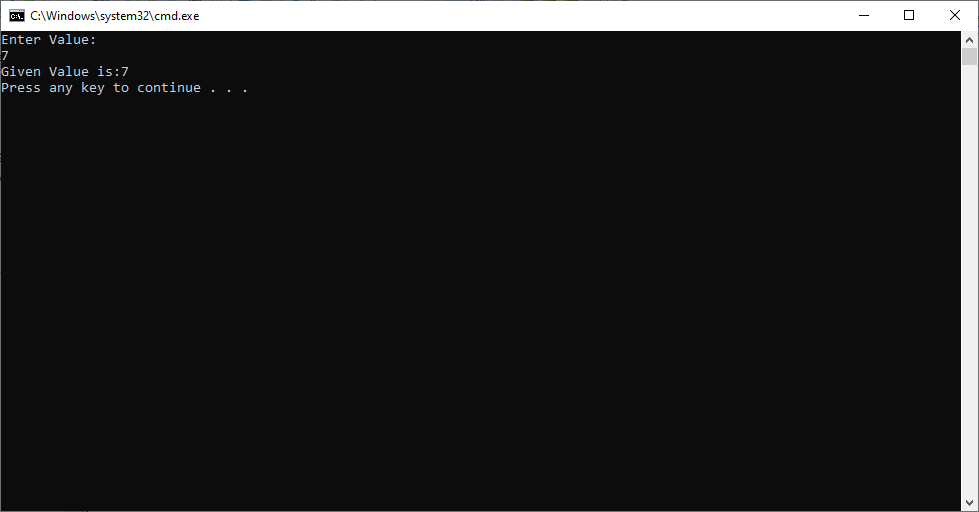
System.out.println("Enter Value:");

char a=(char)System.in.read();

System.out.println("Given Value is:"+ a);

}

}

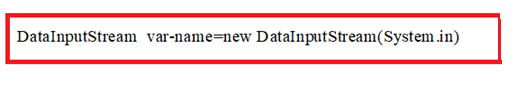


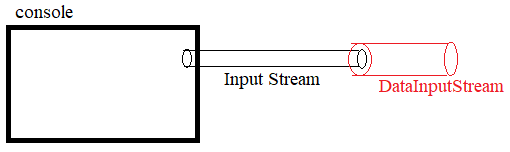
Drawback:- For reading Complete Data(line) that User has given on console, Programmer must write a lot of logic.

The solution is DataInutStream Class.

**10.DataInputStream**:- It is class available “java.io “package. This class object can’t directly be connected to destination(console,file,socket…etc). That’s why It is called as ‘**filter object**.’ we can connect this object to another stream object which can directly be connected to destination(console,log,socket).

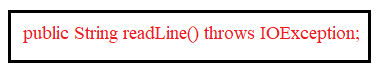
Syntax:





This class has method called readLine() which can read complete line of data that user has entered on console.

Syntax:



Example:1

import java.io.\*;

class sample

{

public static void main(String[] args) throws IOException

{

System.out.println("Enter Integer Value:");

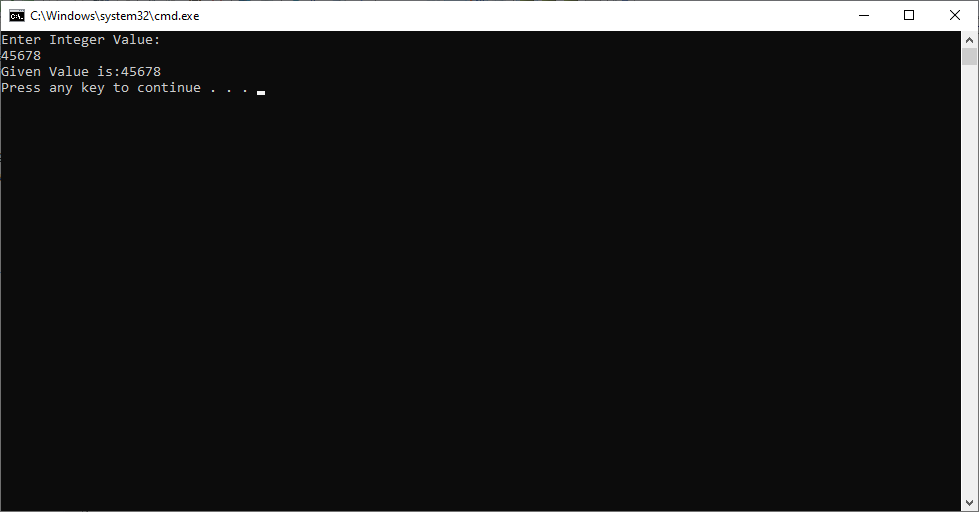
DataInputStream di=new DataInputStream(System.in);

int a=Integer.parseInt(di.readLine());

System.out.println("Given Value is:"+ a);

}

}

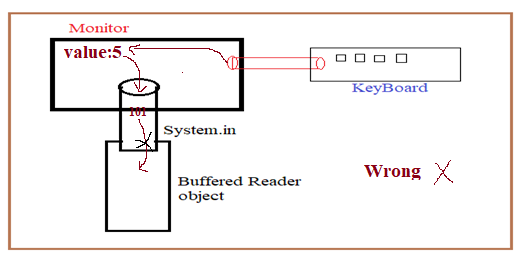


Drawback:- The readLine() method is deprecated tag. Deprecation means that feature is currently available but in upcoming version , the same feature may not be available.

**11.Buffered Reader:**It is class available “java.io “package. This class object can’t directly be connected to destination(console,file,socket…etc). That’s why It is called as ‘**filter object**.’

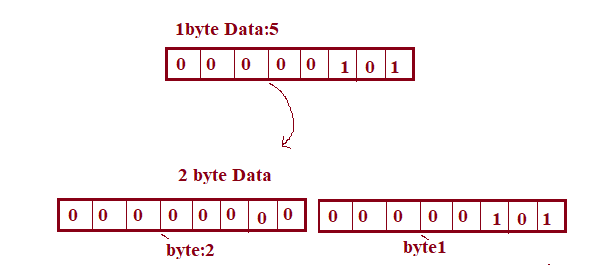
we should not connect this object directly to System.in object. It leads to CE.

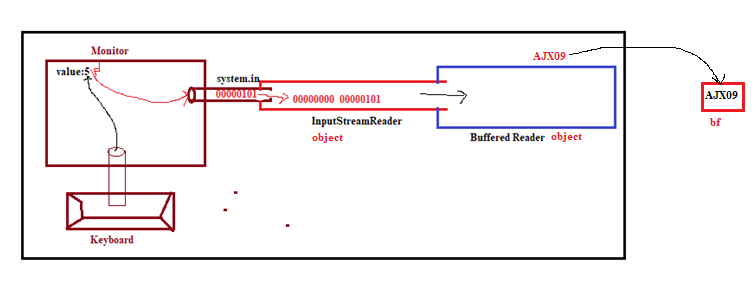
BufferedReader b=new BufferedReader(System.in);



Why CE: InputStream object/Byte Stream(system.in) has 1byte size data. But Buffered Reader object expects 2bytes(character) data. There is gap/mismatch between the System.in object and BufferedReader object.The gap is filled by ‘InputStreamReader’ object.

11.1.InputStreamReader:- It is bridge between byte stream and Character stream. It reads 1byte size data from byte stream and It convert into 2bytes size data.





Syntax to create Buffered Reader class object:



This class has one method to read the entire line from console. Method name is **readLine().**



Example: Develop a program to read data from keyboard and display them on monitor.

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.IOException;

class sample

{

public static void main(String[] args) throws IOException

{

BufferedReader bf=new BufferedReader(new InputStreamReader(System.in));

System.out.print("Enter number:");

int a=Integer.parseInt(bf.readLine());

System.out.print("Enter floatValue:");

float b=Float.parseFloat(bf.readLine());

System.out.print("Enter boolean:");

boolean c=Boolean.parseBoolean(bf.readLine());

System.out.print("Enter String:");

String d=bf.readLine();

System.out.println(a);

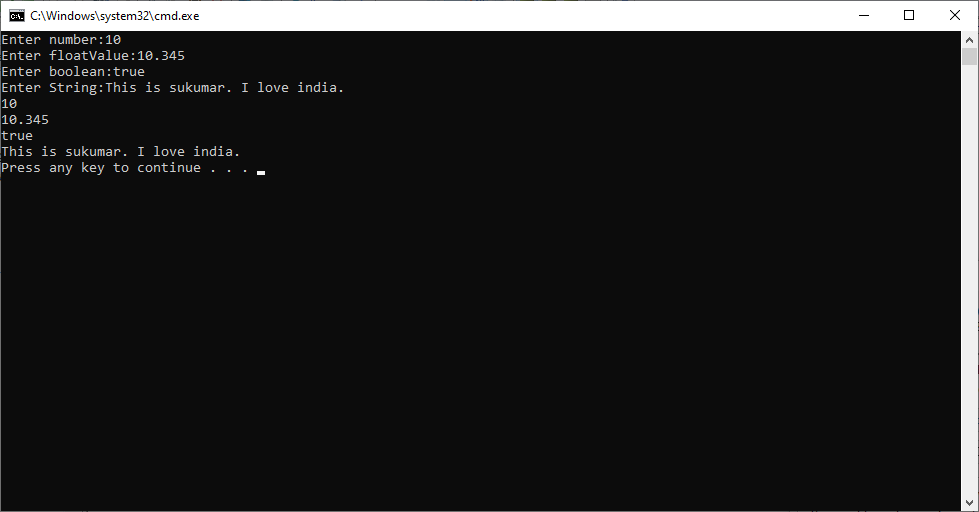
System.out.println(b);

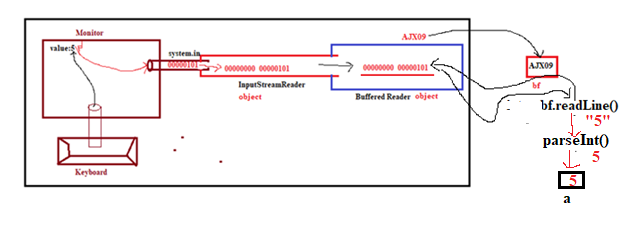
System.out.println(c);

System.out.println(d);

}

}





Drawbacks:-

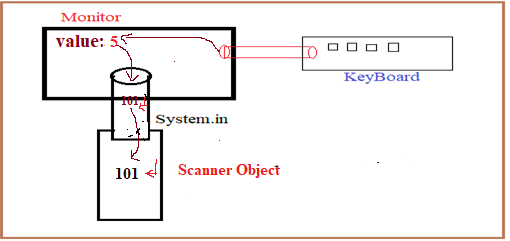
1. Every value that Program has to be converted into specific data type using wrapper class Method.
2. Programmer should write long statement to create BufferedReader Object.
3. IOException should be handled.

To overcome above problems, Sun Micro System gave “Scanner” class in java 5.0 version.

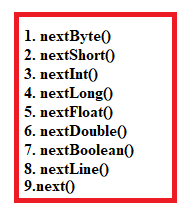
**12.Scanner:** It is class. It is available in ‘java.util” package.

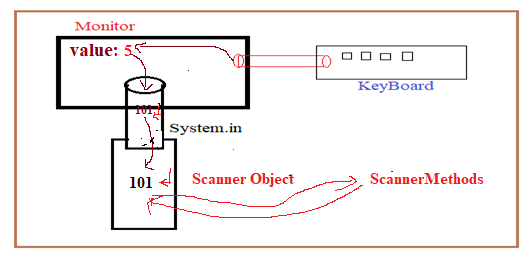
Syntax:





This class has following Methods. These methods read data as string from scanner object and these methods converts it into corresponding datatype data.





**Diff between next() and nextLine() method:-**

The next () method will read only one word. It means it read only the characters available upto space.

NextLine method will read complete line, it means it will read all characters available upto enter key.

When we use next() method with the combination other nextXXX() method, program execution is paused because for the next() method the identification for reading values is space, because there is no space in Scanner object it will pause program execution.

Where as when we use nextLine() method in combination with other nextXXX() methods including next() method program execution is not paused, because it reads the left enter key character.

**Diff between nextLine() and other methods**:-The nextLine() method read all data including ‘enter’ from scanner object. The remaining methods read data upto before the‘space’ or ‘enter’ character from scanner object.

Example:1 Develop a program to read data from keyboard and display them on console.

import java.util.Scanner;

class sample

{

public static void main(String[] args)

{

Scanner scr=new Scanner(System.in);

System.out.print("Enter number:");

int a=scr.nextInt();

System.out.print("Enter floatValue:");

float b=scr.nextFloat();

System.out.print("Enter boolean:");

boolean c=scr.nextBoolean();

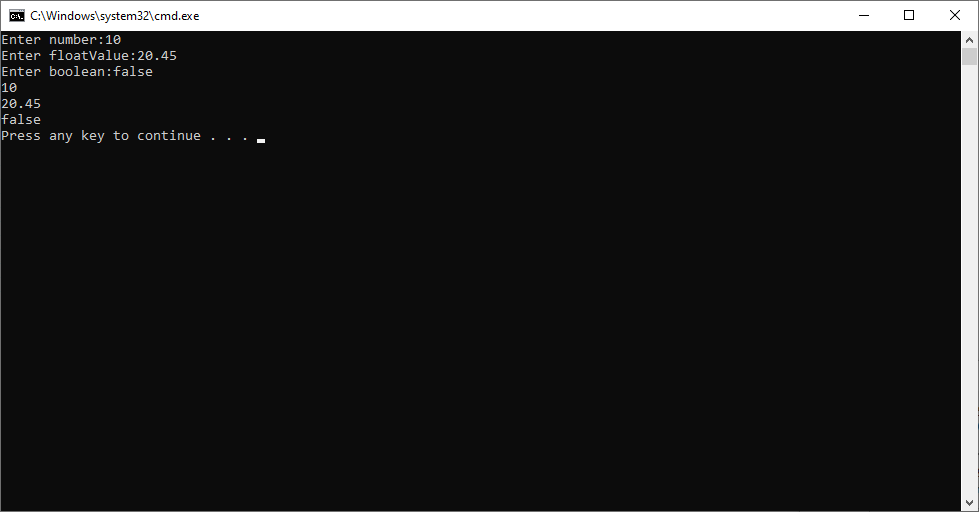
System.out.println(a);

System.out.println(b);

System.out.println(c);

}

}



Example:2 The Program demonstrates important problem.

import java.util.Scanner;

class sample

{

public static void main(String[] args)

{

Scanner scr=new Scanner(System.in);

System.out.print("Enter number:");

int a=scr.nextInt();

System.out.print("Enter String:");

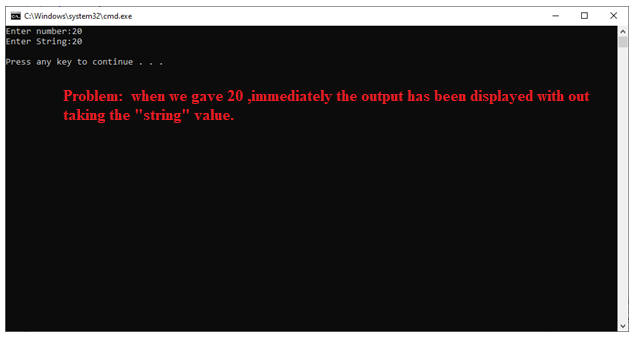
String d=scr.nextLine();

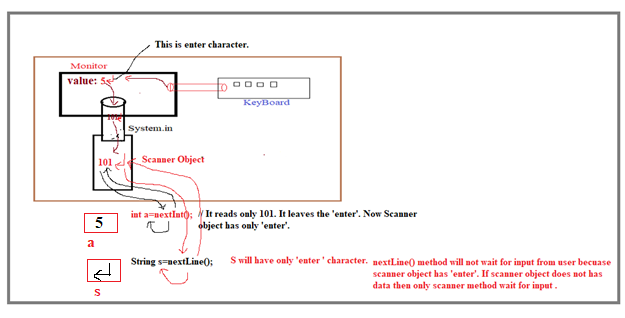
System.out.println(a);

System.out.println(d);

}

}





Problem overcoming:

import java.util.Scanner;

class sample

{

public static void main(String[] args)

{

Scanner scr=new Scanner(System.in);

System.out.print("Enter number:");

int a=scr.nextInt();

scr.nextLine();

System.out.print("Enter String:");

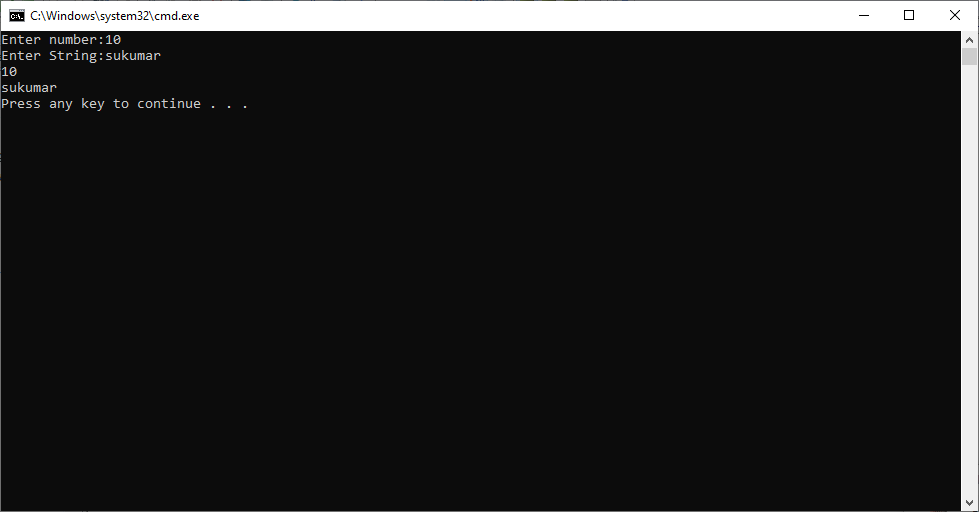
String d=scr.nextLine();

System.out.println(a);

System.out.println(d);

}

}



Example:3

import java.util.Scanner;

class sample

{

public static void main(String[] args)

{

Scanner scr=new Scanner(System.in);

System.out.print("Enter number:");

int a=scr.nextInt();

int b=scr.nextInt();

int c=scr.nextInt();

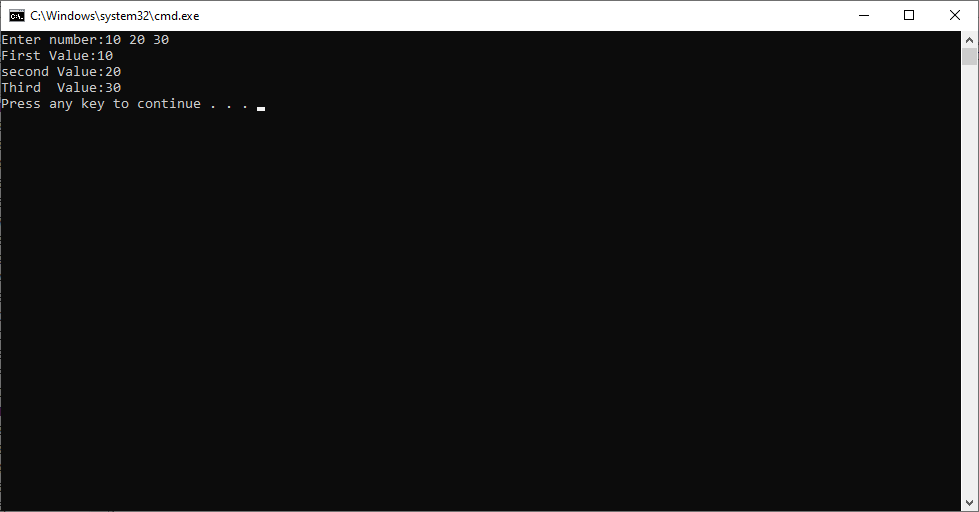
System.out.println("First Value:"+a);

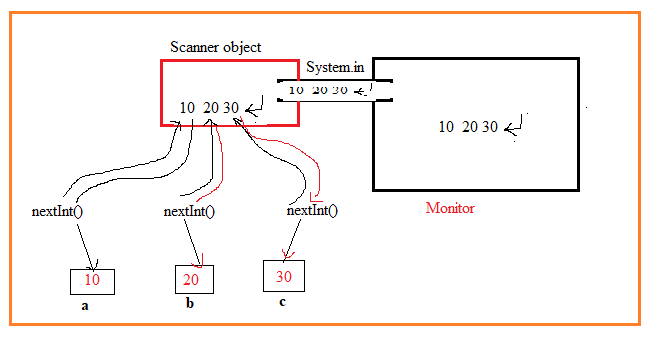
System.out.println("second Value:"+b);

System.out.println("Third Value:"+c);

}

}





Example:4 Develop a program that demonstrates show difference next() and nextLine() method.

import java.util.Scanner;

class sample

{

public static void main(String[] args)

{

System.out.println("===============CASE-1=================================");

Scanner scr=new Scanner(System.in);

System.out.print("Enter Name:");

String a=scr.nextLine();

System.out.print("Enter course:");

String b=scr.next();

System.out.println("Name::"+a);

System.out.println("Course:"+b);

System.out.println("================CASE-2================================");

Scanner scr1=new Scanner(System.in);

System.out.print("Enter Name:");

String c=scr1.next();

System.out.print("Enter course:");

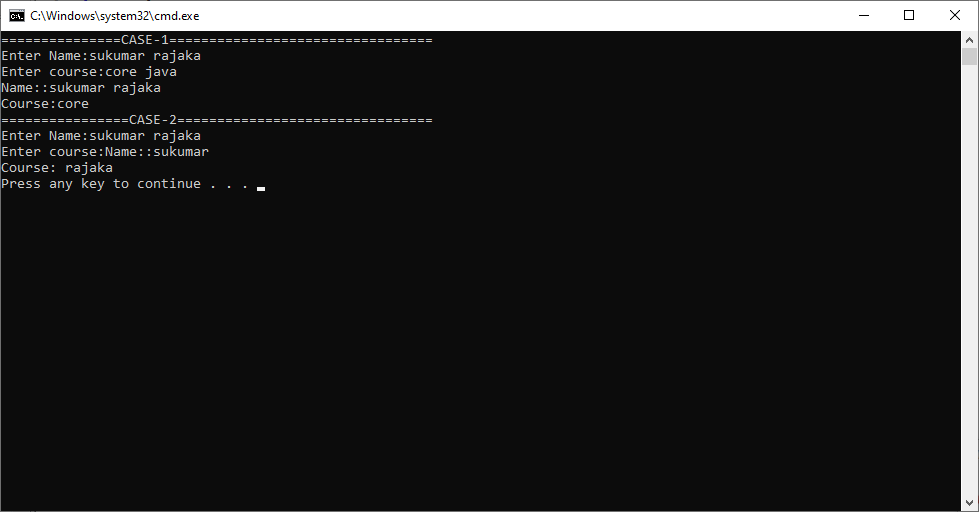
String d=scr1.nextLine();

System.out.println("Name::"+c);

System.out.println("Course:"+d);

}

}



Problem with all above approaches:

In all above approaches we can not hide confidential values . while typing confidential values like password, pin numbers those values are visible to others.

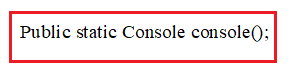
Solution: we must use “Console” class.

**13.Console:**Console is a class given in java 6 version. It is available in “java.lang package”. Using this class , Program read both normal values and confidential values by hiding their characters on console.

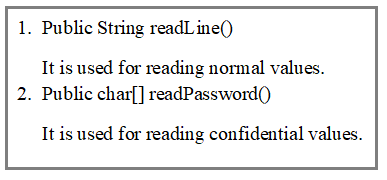
We can not create Console class object by using new keyword & constructor, because Console class contains private Constructor. The reason behind declaring constructor as private specially in Console class is to stop creating new Console object by ourself and for forcing us to obtain existing console object in which out program is loaded and executing.

Note: console means cmd prompt.

The program obtains the Console object by invoking console() factory method. This is a static method and defined in “System” class.



This class has following two methods.



Note:- readPassword() will hide the chracters while we are type on console. It will not shou you even \*\*\*\* or even it will not move cursor to next position while typing chars. While typing password chars cursor will be just blinking on the same position on console.

Example:- Develop a program to read username and password from console. Program must hide password characters.

import java.io.Console;

class sample

{

public static void main(String[] args)

{

Console cn=System.console();

System.out.print("User Name:");

String a=cn.readLine();

System.out.print("Enter Password:");

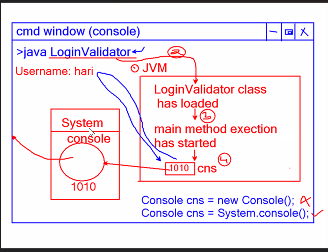
char [] b=cn.readPassword();

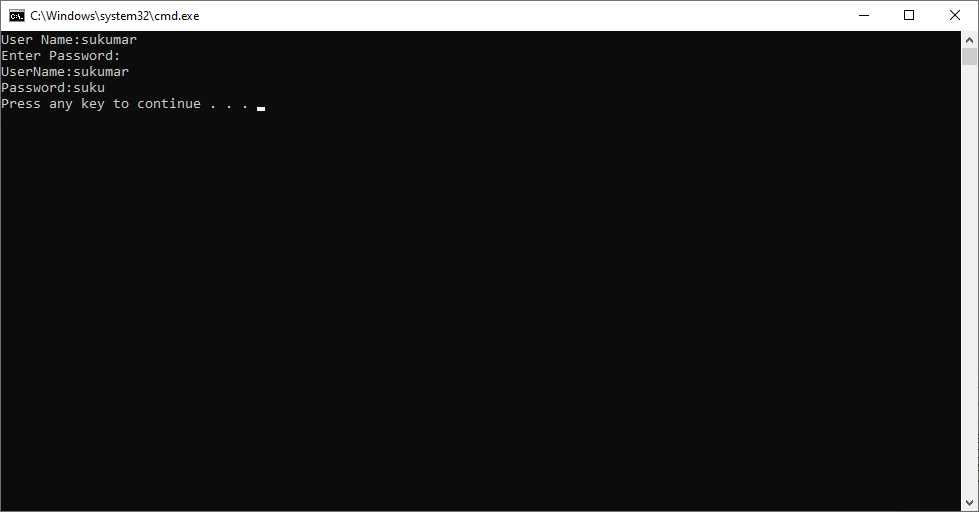
System.out.println("UserName:"+ a);

System.out.println("Password:"+ new String(b));

}

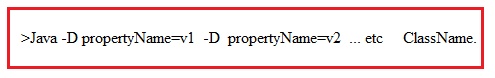
}





14.Working with System Properties:- For storing values at JVM level common to all classes and all projects running in this JVM, we must use System properties approach. In general, System properties approach is used setting JVM configuration values. **We do not use this approach for our project runtime values.** We can set System properties in two ways.

1. from command line by using java -D option.

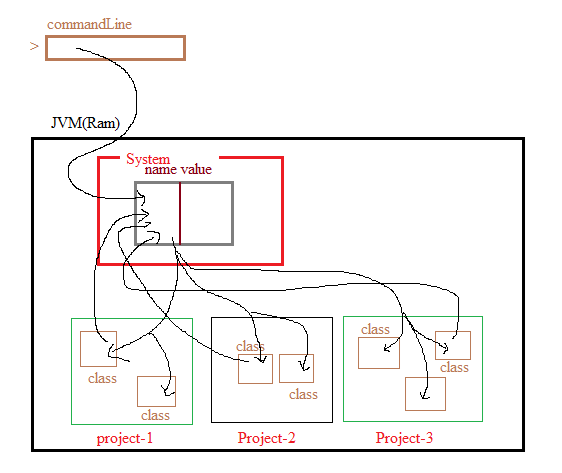


2. from class using System.setProperty(name,value) method.

From class , we can read property values using getProperty(name) method.

Syntax: String getProperty(name);

If property does not exist , this method returns ‘null’.



Example:1

import java.io.Console;

class sample

{

public static void main(String[] args)

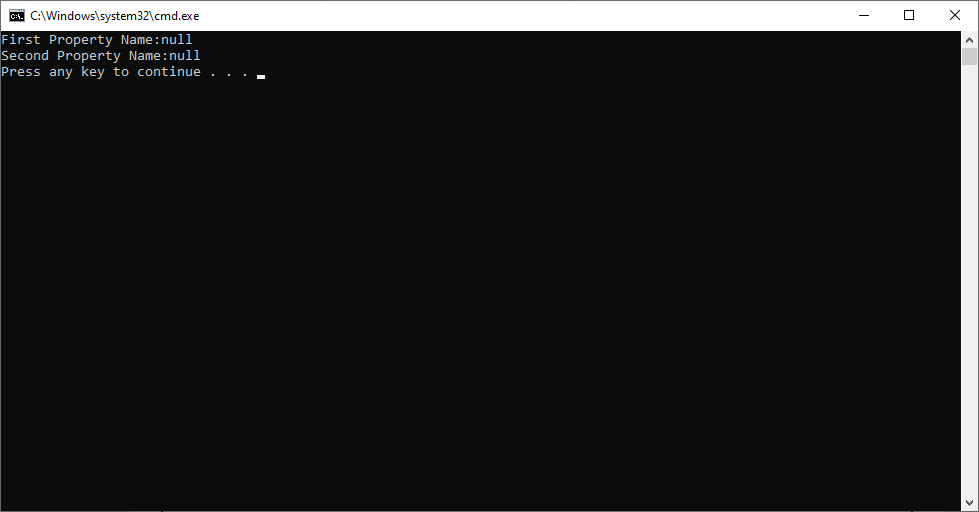
{

System.out.println("FirstProperty:"+ System.getProperty("name"));

System.out.println("SecondProperty:"+ System.getProperty("age"));

}

}



Example:2 Setting item/value from command Line.

import java.io.Console;

class sample

{

public static void main(String[] args)

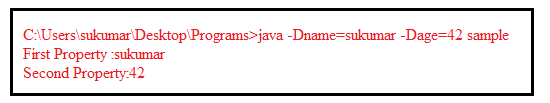
{

System.out.println("FirstProperty:"+ System.getProperty("name"));

System.out.println("SecondProperty:"+ System.getProperty("age"));

}

}



Example:3 Setting item/value from class.

import java.io.Console;

class sample

{

public static void main(String[] args)

{

System.setProperty("name","SV");

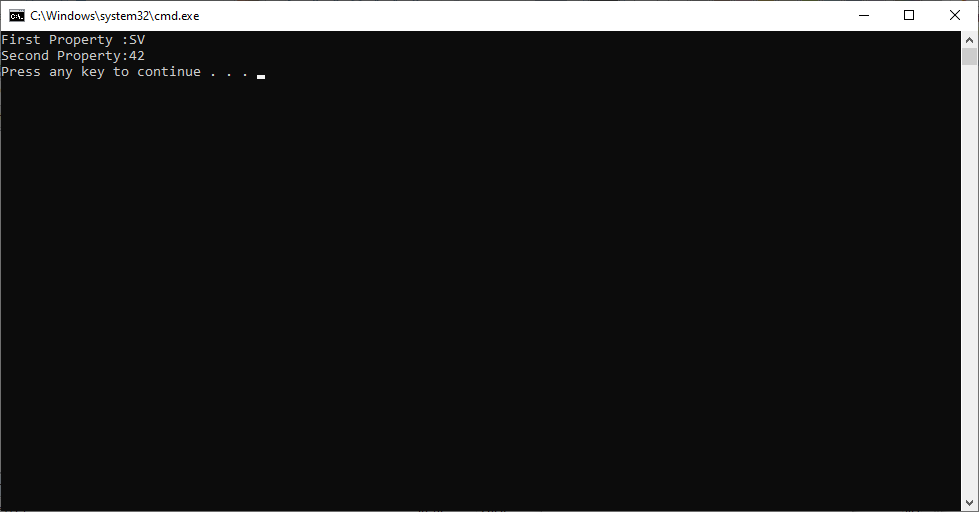
System.setProperty("age","42");

System.out.println("First Property :"+ System.getProperty("name"));

System.out.println("Second Property:"+ System.getProperty("age"));

}

}



EXERCISE

