

```

if (in != -1)
{
response.append(inChar);
}
} while ((in != 1) & (inChar != '\n'));
buff.close();
return response.toString();

}
catch (IOException e)
{
System.out.println("Exception: " + e.getMessage());
return null;
}
}
public static void main(String[] arguments)
{
System.out.print("\nWhat is your name?");
String input = ConsoleInput.readLine();
System.out.println("\nHello, " + input);
}
}

```

### ***Writing Console Output***

we have to use System.out for standard Output in Java. It is mostly used for tracing errors, or for sample programs. These sources can be associated with a writer and sink objects by wrapping them in writer object. For standard output, an OutputStreamWriter object can be used, but this is often used to retain the functionality of print and println methods. In this case, the appropriate writer is PrintWriter. The second argument to PrintWriter constructor requests that the output will be flushed whenever the println method is used. This avoids the need to write explicit calls to the flush method in order to cause the pending output to appear on the screen. PrintWriter is different from other input/output classes as it doesn't throw an IOException. It is necessary to send check Error message, which returns true if an error has occurred. One more side effect of this method is that it flushes the stream.

```
PrintWriter pw = new PrintWriter (System.out, true)
```

### ***Object serialization***

Serialization takes all the data attributes, writes them out as an object, and reads them back in as an object. For an object to be saved to a disk file it needs to be converted to a serial form. An object can be used with streams by implementing the serializable interface. The serialization is used to indicate that objects of that class can be saved and retrieved in serial form. Object serialization is quite useful when you need to use object persistence. By object persistence, the stored object continues to serve the purpose even when no Java program is running, and stored information can be retrieved in a program so it can resume functioning, unlike the other objects that cease to exist when object stops running.

DataOutputStreams and DataInputStreams are used to write each attribute out individually, and then can read them back in at the other end. But to deal with the entire object, not its individual attributes, store away an object or send it over a stream of objects.

### **Transient Keyword**

When an object that can be serialized, we have to consider whether all the instance variables of the object will be saved or not. Sometimes, we have some objects or sub objects which carry sensitive information like a password. If we serialize such objects even if information (sensitive information) is private in that object it can be accessed from outside. To control this we can turn off serialization on a field- by-field basis using the transient keyword.

#### **//Program**

```
import java.io.*;
import java.util.*;
public class SerialDemo implements Serializable
{
    private Date date = new Date();
    private String username;
    private transient String password;
    SerialDemo(String name, String pwd)
    {
        username = name;
        password = pwd;
    }
    public String toString()
    {
        String pwd = (password == null) ? "(n/a)" : password;
        return "Logon info: \n " + "Username: " + username +
            "\n Date: " + date + "\n Password: " + pwd;
    }
    public static void main(String[] args)
        throws IOException, ClassNotFoundException
    {
        SerialDemo a = new SerialDemo("Java", "sun");
        System.out.println( "Login is = " + a);
        ObjectOutputStream o = new ObjectOutputStream( new
            FileOutputStream("Login.out"));
        o.writeObject(a);
        o.close();
        // Delay:
        int seconds = 10;
        long t = System.currentTimeMillis()+ seconds * 1000;
        while(System.currentTimeMillis() < t)
        ;
        // Now get them back:
        ObjectInputStream in = new ObjectInputStream(new
            FileInputStream("Login.out"));
```

```

System.out.println("Recovering object at " + new Date());
a = (SerialDemo)in.readObject();
System.out.println( "login a = " + a);
}

}

```

### **Volatile Modifier**

The volatile modifier is used when we are working with multiple threads. The Java language allows threads that access shared variables to keep private working copies of the variables. This allows for a more efficient implementation of multiple threads. These working copies need to be reconciled with the master copies in the shared (main) memory only at prescribed synchronization points, namely when objects are locked or unlocked. As a rule, to ensure that shared variables are consistently and reliably updated, a thread should ensure that it has exclusive use of such variables by obtaining a lock and conventionally enforcing mutual exclusion for those shared variables. Only variables may be volatile. Declaring them so indicates that such methods may be modified asynchronously.

## **USING NATIVE METHODS**

**Native Method** is a method which is not written in Java, and is outside of the JVM in a library. This feature is not special to Java. Most languages provide some mechanism to call routines written in another language. In C++, you must use the extern “C” statement to signal that the C++ compiler is making a call to C functions.

To declare a native method in Java, a method is preceded with native modifiers much like we use the public or static modifiers, but don’t define any body for the method, but simply place a semicolon in its place.

### **Syntax :**

```
native void getdata();
```

### **Q. Differentiate between call-by-value and call-by-reference with the help of suitable example.**

**Ans :** Both call by value and call by reference is used to pass the argument in a function.

**Call by value->** This method copies the value of an argument into the format parameter of the subroutine. Therefore, changes made to the parameter of the subroutine have no effect on the argument.

```

class Test
{
    void call(int i,int j)
    {
        i=i*2;
        j=j/2;
    }
}

```

```

class callbyvalue
{
    public static void main(String args[])
    {
        Test ob=new Test();
        int a=15;
        int b=20;
        System.out.println(" a and b before call" +a + " " +b);
        ob.call(a,b);
        System.out.println("a and b after call" +a + " " +b);
    }
}

```

**Call by reference** -> In this method, a reference to an argument is passed to the parameter. Inside the subroutine, this reference is used to access the actual argument specified in the call. This means that changes made to the parameter will effect the argument used to call the subroutine.

```

class Test
{
    int a,b;
    test(int i,int j)
    {
        a=i;
        b=j;
    }
    void call(Test o)
    {
        o.a*=2;
        o.b/=2;
    }
}
class callbyref
{
    public static void main(String args[])
    {
        Test ob=new Test(15,20);
        System.out.println(" a and b before call" +ob.a + " " +ob.b);
        ob.call(ob);
        System.out.println("a and b after call" +ob.a + " " +ob.b);
    }
}

```

### **Q. Two methods used for applet execution**

**Ans :** The two methods for executing an applet in java are :

- i) One using internet explorer and using HTML file.
- ii) Second using class AppletViewer.

### **Q. Write a recursive function in java to print the gcd of two given integers.**

**Ans :** public class gcd

```

{
    public static void main(String args[])

```

```

    {
        int a, b;
        //User input for both numbers
        a=Integer.parseInt(args[0]);
        b=Integer.parseInt(args[1]);

        System.out.println("The gcd of "+a+" and "+b+" is "+gcd(a,b)+".");
    }
    static int gcd(int a,int b)
    {
        if(b==0) return a;
        else return gcd(b,a%b);
    }
}

```

**Q. Static function can only use static data of the class, justify your answer.**

**Ans :** static variables is the variable which use to have static value for all the class. It is common for all the instance of the class. When an instance variable is declared static, then at the time of instantiation of class, it is initialize.

If one want to initialize all the static variables, then a member of static type can be created. Since the method is static, so one can access it without instantiation of class also.

**e.g,** class xyz

```

{
    static int n1=5;
    static int n2;
    static void init(int x)
    {
        System.out.println("n1=" +n1);
        System.out.println("n2=" +n2);
        System.out.println("x=" +x);
    }
    public static void main(String args[])
    {
        init(10);
    }
}

```

**Q. What are the points to ensure while writing a method that is intended to override another method.**

**Ans :** Some important points that must be taken care while overriding a method.

- An overriding method replaces the method it overrides.
- Each method in a parent class can be overrides at most once in any of the subclass.
- Overriding methods must have exactly same arguments lists, both in type and in order.
- an overriding method must have same return type as the method it overrides.
- Overriding is associated with inheritance.

**Q. Differentiate between interfaces and abstract classes with the help of suitable examples.**

**Ans :** Abstract class and interfaces similarity between them that methods of both should be implemented but there are many differences between them these are :

- i) A class can implement more than one interface, but abstract class can only subclass one class.
- ii) An abstract class can have non-abstract methods. All methods of an interface are implicitly or explicitly abstract.
- iii) An abstract class can declare instance variable but an interface can't.
- iv) Every method of interface is implicitly public.

**Q. What is data hiding? Explain with example.**

**Ans :** The wrapping of data and functions into a single unit is known as encapsulation and unit is known as class. The data is not directly accessible to the outside world and only in functions, which are wrapped in the class, can access it. Functions are made available to outside world. These functions provide an interface to access data. If one wants to modify data of an object. It should be known exactly the functions that are available to interact with it. This insulation of data from direct access by program is known as data hiding. E.g,

```
class Area
{
    private int a,b;
    public void calArea(int x,int y)
    {
        a=x;
        b=y;
        return a*b;
    }
}
```

**Q. Write a recursive function in java to print a given string in reverse order.**

```
Ans : import java.io.*;
class revorder
{
    public static void main(String args[]) throws IOException
    {
        BufferedReader ob=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("type in a statement you want to reverse");
        String st=ob.readLine();
        String revstr="";
        int len=st.length();
        char sp=' ';
        for(int i=len;i>=0;i--)
        {
            if(st.charAt(i)==sp)
            {
                for(int j=len;j>=0;j--)
                {
                    revstr=revstr+st.charAt(j);
                }
            }
        }
        System.out.println(revstr);
    }
}
```

## **1. What is java Platform?**

**Ans :** When Java Code is compiled a byte code is generated which is independent of the system. This byte code is fed to the JVM (Java Virtual Machine) which resides in the system. Since every system has its own JVM, it doesn't matter where we compile the source code. The byte code generated by the compiler can be interpreted by any JVM of any machine. Hence it is called Platform independent Language.

Java's byte code are designed to be read and interpreted in exactly same manner on any computer hardware or operating system that supports Java Runtime Environment.

When we write the program for any programming language it is called (source program), and it will have the extension depending upon the language.

In Java, when we write the program, we will have the ".java" file. And when it is compiled, we will get the ".class" file. The ".class" file is executed by the Java Virtual Machine (JVM). If we have the JVM, we can execute the java program anywhere under operating system. That means, that Java is PLATFORM INDEPENDENT.

## **Q. What is a default layout manager in java. Explain it.**

### **i) Border Layout**

The border layout is the default layout manager for all Window objects. It consists of five fixed areas: North, South, East, West, and Center. You do not have to put a component in every area of the border layout. The demonstration puts the label "Border Layout" in the North area, the OK button in the South area, and the List of fruit trees in the Center area. The East (right) and West (left) areas are empty.

If any or all of the North, South, East, or West areas are left out, the Central area spreads into the missing area or areas. However, if the Central area is left out, the North, South, East, or West areas do not change.

When adding components, we will use these constants with the following form of add(), which is defined by Container.

Void add( Component compObj, Object region);

Here, compObj is the component to be added, and region specifies where the component will be added.

### **ii) FlowLayout**

The flow layout is the default layout manager for all Panel objects and applets. It places the panel components in rows according to the width of the panel and the number and size of the components. FlowLayout implements a simple layout style, which is similar to how words flow in a text editor. Components are laid out from the upper-left corner, left to right and top to bottom. When no more components fit on a line, the next one appears on the next line.

**1. Explain any six bitwise operators of Java. Also, demonstrate each operator with an example.**

**Ans :** There are different operations can be performed using bitwise operators:

**a) bitwise AND->** The bitwise AND operation will be carried out between the two bit patterns of the two operands. For example,

**Truth Table of AND**

A	B	F (Output)
0	0	0
0	1	0
1	0	0
1	1	1

**Example**

**Binary pattern**  
i) int x = 5;    0101  
   int y = 2;    0010  
int z = x & y;   0000

so, z = 0

**b) bitwise OR->** The bitwise OR operations are similar to the bitwise AND and the result is 1 if any one of the bit value is 1. The symbol | represents the bitwise OR.

**Truth Table of OR**

A	B	F (Output)
0	0	0
0	1	1
1	0	1
1	1	1

**Example :**

**Binary Pattern**  
i) int x = 5;    0101  
   int y = 2;    0010  
  
int z = x | y    0111

so, z = 7



**c) bitwise exclusive OR**-> The bitwise exclusive OR will be carried out by the notation  $\wedge$ . To generate a 1 bit in the result, a bitwise exclusive OR needs a one in either number but not in both.

#### Truth Table of XOR

A	B	F (Output)
0	0	0
0	1	1
1	0	1
1	1	0

#### Example :

	<b>Binary Pattern</b>		<b>Binary Pattern</b>
i) int x = 5;	0101	ii) int a = 6	0110
int y = 2;	0010	int b = 3	0011
int z = x $\wedge$ y	0111	int c = a $\wedge$ b	0101

so, z = 7

so, c = 5

**d) Shift Operations** -> The shift operations take binary patterns and shift the bits to the left or right, keeping the same number of bits by dropping shifted bits off the end and filling in with zeros from the other end. Java provides two types of shift operations, left shift and right shift.

**i) left shift** -> The << operator is used for left shifting.

#### Example :

<b>Variable</b>	<b>Value</b>	<b>Binary Patterns</b>
i) X	33	0010 0001(8 bits)
X<<1		- the bit pattern of the X value is left shifted once.

Output : 0100 0010  
So, result is : 66

ii) int X = 33	0010 0001(8 bits)
X<<3	the bit pattern of the X value is left shifted by thrice.
	0 0100 0010
	0 1000 0100
	1 0000 1000

Since, The resultant is 8 bit pattern  
So, the resultant bit pattern will be : 0000 1000  
So, result is : 8

ii) **right shift** -> The right shift >> operator is used for right shifting.

**Example :**

	<b>Binary Pattern</b>
i) <b>y = 41,</b>	0010 1001
<b>y&gt;&gt;3</b>	the bit pattern of the y value is right shifted by thrice.
00101001	
00010100	1
00001010	0
00000101	0

So, the resultant bit pattern will be : 00000101

So, result is : 5

e) **Bitwise Complement**-> The complement operator ~ switches all the bits in a binary pattern, that is, all the zeroes become ones and all the ones become zeroes. The complement of a pattern is often useful in signaling and controlling other devices where several different signals may be complementary to each other.

**Example :**

<b>Variable</b>	<b>Value</b>	<b>Binary Pattern</b>
X	23	0001 0111(8 bits)
~X	132	1110 1000
Y	ff	1111 1111
~Y	00	0000 0000

### **Q. What is JVM. Explain**

A **Java Virtual Machine (JVM)** enables a set of computer software programs and data structures to use a virtual machine model for the execution of other computer programs and scripts. The model used by a JVM accepts a form of computer intermediate language commonly referred to as Java bytecode.

A JVM can also implement programming languages other than Java. For example, Ada source code can be compiled to Java byte code, which may then be executed by a JVM. JVMs can also be released by other companies besides Oracle (the developer of Java) — JVMs using the "Java" trademark may be developed by other companies as long as they adhere to the JVM specification published by Oracle and to related contractual obligations.

Java was conceived with the concept of WORA: "write once, run anywhere". This is done using the Java Virtual Machine. The JVM is the environment in which java programs execute. It is software that is implemented on non-virtual hardware and on standard operating systems.

JVM is a crucial component of the Java platform, and because JVMs are available for many hardware and software platforms, Java can be both middleware and a platform in its own right, hence the trademark write once, run anywhere. The use of the same byte code for all platforms allows Java to be described as "compile once, run anywhere", as opposed to "write once, compile anywhere", which describes cross-platform compiled languages.

**Q. Write a java program using class and constructor to find the length of string.**

```
import java.lang.reflect.*;

public class DumpMethods {
    public static void main(String args[])
    {
        try {
            Class c = Class.forName(args[0]);
            Method m[] = c.getDeclaredMethods();
            for (int i = 0; i < m.length; i++)
                System.out.println(m[i].toString());
        }
        catch (Throwable e) {
            System.err.println(e);
        }
    }
}
```

**Q. Explain String and StringBuffer class. Explain various string functions.**

Java provides the StringBuffer and String classes, and the String class is used to manipulate character strings that cannot be changed. Simply stated, objects of type String are read only and immutable. The StringBuffer class is used to represent characters that can be modified.

The significant performance difference between these two classes is that StringBuffer is faster than String when performing simple concatenations. In String manipulation code, character strings are routinely concatenated. Using the String class, concatenations are typically performed as follows:

```
String str = new String ("Stanford ");
str += "Lost!!";
```

If we were to use StringBuffer to perform the same concatenation, we would need code that looks like this:

```
StringBuffer str = new StringBuffer ("Stanford ");
str.append("Lost!!");
```

### **Various string functions**

#### **1. public String concat(String s)**

This method returns a string with the value of string passed in to the method appended to the end of String which used to invoke the method.

```
String s="abcdefg";
System.out.println(s.concat("hijkl"));
```

```
s.concat("hijkl");  
System.out.println(s);
```

## **2. public charAt(int index)**

This method returns a specific character located at the String's specific index. Remember, String indexes are zero based. Example

```
String s="Alfanzo Mango";  
System.out.println(s.charAt(0));
```

The output is 'A'

## **3. public int length()**

This method returns the length of the String used to invoke the method. example:-

```
String s="name";  
System.out.println(s.length());
```

The output is 4

## **4. public String replace(char old, char new)**

This method return a String whose value is that of the String to invoke the method ,updated so that any occurrence of the char in the first argument is replaced by the char in the second argument .Example:-

```
String s="VaVavavav";  
System.out.println(s.replace('v','V'));
```

The output is VaVaVaVaV

## **5. public boolean equalsIgnoreCase(String s)**

This method returns a boolean value depending on whether the value of the string in the argument is the same as the String used to invoke the method. This method will return true even when character in the string object being compared have different cases. Example:-

```
String s="Vaibhav";  
System.out.println(s.equalsIgnoreCase("VAIBHAV"));
```

The output is true

## **6. public String substring(int begin) / public String substring(int begin,int end)**

substring method is used to return a part or substring of the String used to invoke the method. The first argument represents the starting location of the substring. Remember the indexes are zero based. example:-

```
String s="abcdefghi";
System.out.println(s.substring(5));
System.out.println(s.substring(5,8));
```

The output would be

```
" fghi "
" fg "
```

### **7.public String toLowerCase()**

This method returns a string whose value is the String used to invoke the method, but with any uppercase converted to lowercase.:-

```
String s="AbcdefghiJ";
System.out.println(s.toLowerCase());
```

Output is " abcdefghij "

### **8.public String trim()**

This method returns a String whose value is the String used to invoke the method ,but with any leading or trailing blank spaces removed.Example:-

```
String s="hey here is the blank space ";
System.out.println(s.trim())
```

The output is " heyhereistheblankspace"

### **Q. What is labeled break in java. Explain.**

1. The break statement can be followed by a label.
2. The presence of a label will transfer control to the start of the code identified by the label.

```
public class Simple
{
    public static void main(String[] args)
    {
        OuterLoop: for (int i = 2;i<=50; i++)
        {
            for (int j = 2; j < i; j++) {
                if (i % j == 0) {
                    continue OuterLoop;
                }
            }
            System.out.println(i);
            if (i == 37) {
                break OuterLoop;
            }
        }
    }
}
```