INDEX

DATA-TYPES

1. What are primitive types in java?

byte, short, int, long, float, double, char, boolean.

2. What are tokens? Name the 5 types of tokens available in Java with an example each.

The smallest individual unit in a program is known as a token. The five types of tokens are- :

a) Keywords:- public, static, void, import etc. (Java's own terms)

b) Identifiers:- (variables) alphabets, digits, under score, dollar sign etc.

c) Literals:- (Constants) integer literals, floating literals, boolean literals, character literals, string literal, the null literal.

d) Punctuators (separators):- () {} [] ; ,

e) Operators = + - < >

3. What are Variables in JAVA?

A variable is a name for a location in memory used to store a data value.

• We use variables to save and restore values or the results of calculations.

• The programmer has to tell Java what type of data will be stored in the variable's memory location. Its type cannot change.

• During the program execution the data saved in the memory location can change; hence the term "variable".

4. Write a note on Variable Declaration?

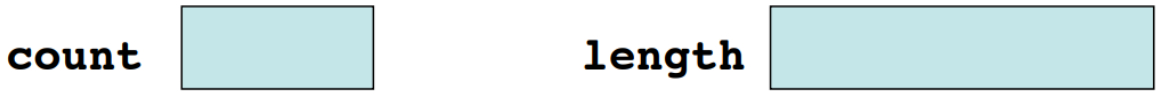
Before you can use a variable, you must declare its type and name.

• You can declare a variable only once in a method.

• Examples:

```
int numDimes;
double length;
char courseSection;
boolean done;
String lastName;
```

Declaring a variable instructs the compiler to set aside a portion of memory large enough to hold data of that type.
int count;
double length;



No value has been put in memory yet. That is, the variable is undefined.

5. Write a note on Declaration and Initialization?

Variables can be declared and initialized in one statement:
Example:

```
int numDimes = 4;
double length = 52.3;
char courseSection = 'J';
boolean done = true;
String lastName = "Reid-Miller";
int count = 3 + 2;
```

6. Write a note on Expressions

An expression is anything that results in a value.
It must have a type.

Arithmetic operators:

Operator	Meaning	Example	Result
+	addition	1 + 3	4
-	subtraction	12 - 4	8
*	multiplication	3 * 4	12
/	division	2.2 / 1.1	2.0
%	modulo (remainder)	14 % 4	2

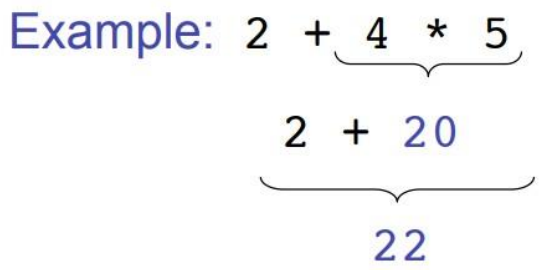
Division and Modulo

```
int a = 40;          double x = 40.0;
int b = 6;           double y = 6.0;
int c;               double z;
```

`c = a / b;`	6	`c = a % b;`	4
`z = x / y;`	6.66666667	`c = b % a;`	6
`c = b / a;`	0	`c = 0 % a;`	0
`z = y / x;`	0.15	`c = b % 0;`	error
`c = 0 / a;`	6		

7. Write a note on OPERATOR PRECEDENCE

The operators *, /, % are evaluated before the operators +, - because *, /, % have higher precedence than +, -.



To change the order use parentheses:
Example: (2 + 4) * 5 evaluates to 30

8. Write a note on evaluating expressions.

When an expression contains more than one operator with the same level of precedence, they are evaluated from left to right.

- `2 + 2 + 3 - 1` is `(((2 + 2) + 3) - 1)` which is `6`
- `2 * 4 % 5` is `((2 * 4) % 5)` which is 3

- `2 * 3 - 2 + 7 / 4`
  `6 - 2 + 7 / 4`
  `6 - 2 + 1`
  `4 + 1`
  `5`

9. Write a note on INTEGER DATA TYPES IN JAVA

An integer is a whole number — that is, a number with no fractional or decimal portion. Java has four integer types, which you can use to store numbers of varying sizes.

Type	Number of Bytes	Range of Values
`byte`	1	–128 to +127
`short`	2	–32,768 to +32,767
`int`	4	–2 billion to +2 billion
`long`	8	–4,000 trillion to +4,000 trillion

The most commonly used integer type is int. You can use short or even byte when you know the variable won't need to store large values, and you can use long when your program will require large values — for example, when calculating the federal deficit.

```
int xInt;
long yLong;
xInt = 32;
yLong = xInt;
```

Java does not allow the converse, however. The following code is not valid:

```
int xInt;
long yLong;
yLong = 32;
xInt = yLong;
```

10. Write a note on char data type in java

In Java, char is short for character. It is 16 bits in size, double that of a byte. Most of the time however, the actual data stored in the char data type doesn't take up more than 8 bits; the reason Java allows 16 bits is so that all characters in all languages can be represented. This representation is in the Unicode format.
Unicode is a computer encoding methodology that assigns a unique number for every character. It doesn't matter what language, or computer platform it's on. This is important in a global, networked world, and for computer systems that must accommodate multiple languages and special characters. Unicode truly unifies all of these into a single standard.

In Unicode, all characters are represented by numeric values. For example, 65 is the letter A. 66 is B and so on. The lower-case letters start at 97. There are even special system-only characters in the list: carriage returns, tabs, etc. These can be useful in displaying text; while others are leftovers from older systems.
Examples: Declaring Char Variables
You'd think that a char could be any value from a to Z, and all the numbers. That is true, except for one key item. You can either declare a char variable with single quotes, e.g., setting a char's value to the letter a. Or, you could omit the quotes, and set the Unicode representation of the value. Take a look at the following code for declaring a char variable equal to 77.

```
char eighty_eight = 77;
```

The output is not going to be 77, but the Unicode representation

11. Write a note on Floating Point Data type in Java

Floating-point numbers are numbers that have fractional parts (usually expressed with a decimal point). You should use a floating-point type in Java programs whenever you need a number with a decimal, such as 19.95 or 3.1415.

Java has two primitive types for floating-point numbers:
float: Uses 4 bytes
double: Uses 8 bytes
In almost all cases, you should use the double type whenever you need numbers with fractional values.
The precision of a floating-point value indicates how many significant digits the value can have following its decimal point. The precision of a float type is only about six or seven decimal digits, which isn't sufficient for most types of calculations.
If you use Java to write a payroll system, for example, you might get away with using float variables to store salaries for employees such as teachers or fire fighters, but not for professional baseball players or corporate executives.
By contrast, double variables have a precision of about 15 digits, which is enough for most purposes.
Example:
double period = 99.0;
If you omit the decimal point, the Java compiler treats the literal as an integer. Then, when it sees that you're trying to assign the integer literal to a double variable, the compiler converts the integer to a double value. This avoidable conversion step uses some precious processing time.

To save that time, you can add an F or D suffix to a floating-point literal to indicate whether the literal itself is of type float or double. For example:
float value1 = 199.33F;
double value2 = 200495.995D;
If you omit the suffix, D is assumed. As a result, you can usually omit the D suffix for double literals.

Type	Size	Range of values that can be stored
float	4 bytes	3.4e−038 to 3.4e+038
double	8 bytes	1.7e−308 to 1.7e+038

12. Write a note on Boolean data type in java.

Boolean is the primitive data type in Java. There are only two values that a boolean type can take they are: true or false. Boolean type is used when we want to test a particular condition during the execution of the program. Boolean values are often used in Selection statements and Iteration statements.

Boolean type is denoted by the keyword boolean and their size is only '1-bit' and not '1-byte'. The words true and false cannot be used as identifiers. Boolean is also the type required by conditional expression that govern the Control statements such as if condition and for loop.

Example:

```
boolean val=true;
```

ARRAYS

13. What do you mean by an Array?

Array is a set of similar data type.
Arrays objects store multiple variables with the same type.
It can hold primitive types and object references.
Arrays are always fixed

14. How to create an Array?

An Array is declared similar to how a variable is declared, but you need to add [] after the type.

Example: int []a;
We can declare Java array as a field, static field, a local variable, or parameter, like any other variable. An array is a collection of variables of that type. Here are a few more Java array declaration examples:
String [] sArray;
MyClass [] myClassArray;

15. What are the advantages and disadvantages of an array.

Advantages:
1. We can put in place other data structures like stacks, queues, linked lists, trees, graphs, etc. in Array.
2. Arrays can sort multiple elements at a time.
3. We can access an element of Array by using an index.

Disadvantages:
1. We have to declare Size of an array in advance. However, we may not know what size we need at the time of array declaration.
2. The array is static structure. It means array size is always fixed, so we cannot increase or decrease memory allocation.

16) Can we change the size of an array at run time?

No we cannot change the array size.

17. Can you declare an array without assigning the size of an array?

No, we cannot declare an array without assigning size.
If we declare an array without size, it will throw compile time error
Example: marks = new int []; //COMPILER ERROR

18. What is the default value of Array?

Any new Array is always initialized with a default value as follows
For byte, short, int, long – default value is zero (0).
For float, double – default value is 0.0.
For Boolean – default value is false.
For object – default value is null.

19. How to print element of Array?

int schoolmarks [] = {25, 35, 55, 10, 5 };
System.out.println (Arrays.toString (schoolmarks));
Output is: [25, 35, 55, 10, 5]

20. How do you compare Two Arrays?

If two arrays are of the same size & data type then comparison can be done by using Arrays.equals ()
Example:
int [] num1 = { 1, 2, 3 };
int[] num2 = { 4, 5, 6 };
System.out.println (Arrays. equals (num1, num2)); //output:false
Int [] num3 = {1, 2, 3};
System.out.println (Arrays.equals (num1, num3)); //output:true

21. How to sort an Array?

Sorting of an array is possible using the inbuilt static method "Arrays.sort()"

Example: int M [] = {12, 5, 7, 9};
Arrays.sort(M);
System.out.println(Arrays.toString(M));//[5, 7, 9, 12]

22. Can we declare array size as a negative number?

No. We cannot declare the negative integer as an array size.
If we declare, there will be no compile-time error.
However, we will get NegativeArraySizeException at run time

23. When will we get ArrayStoreException?

It is a runtime exception. For example, we can store only string elements in a String Array. If anybody tries to insert integer element in this String Array, then we will get ArrayStoreException at run time.

24. Can we add or delete an element after assigning an array?

No it is not possible.

25. What do you mean by an anonymous array? Explain with an example?

Anonymous array means array without any reference.
Example:

```
//Creating anonymous arrays
System.out.println(new int[]{11, 12, 13, 14, 15,16}.length); //Output : 6
System.out.println(new int[]{31, 94, 75, 64, 41,30}[1]); //Output : 94 }}
```

26. Is there any difference between int[] a and int a[]?

No difference both are the legal statement

27. There are 2 integer arrays. One containing 50 elements and other containing 30 elements. Can we assign the array of 50 elements to an array of 30 elements?

Yes we can assign provided they should the same type. The compiler will check the only type of the array, not the size.
Example:

```
int[] a = new int[30];
int[] d = new int[50];
a = d; //Compiler checks only type, not the size
```

28. int a[] = new int[3]{1, 2, 3} – is it a right way to declare arrays in java?

No, we should not mention the size of the array when we are providing the array elements.

29. How to copy one array into another array?

Below four tricks are available in java to copy an array.
Using "For loop"
Using "Arrays.copyOf()" method
Using "System.arraycopy()" method
Using "clone()" method

30. What are "jagged" arrays in java?

Jagged Arrays are Arrays that contains arrays of different length. Jagged arrays are also known as multidimensional arrays.

31. When ArrayIndexOutOfBoundsException occurs?

It will occur when the program tries to access invalid index of an array. Index higher than the size of the array or negative index.

32. How do we search a specific element in an array?

We can use Arrays.binarySearch() method. This method uses binary search algorithm.

33. What would happen if you do not initialize an array?

Array will have default value.

34. How do we find duplicate elements in an array?

Using Brute Force Method:
In this method, we compare each element of an array with other elements.
If any two elements found equal or same, we declare them as duplicates.
Example:

```
public class DuplicatesInArray
 {
```

```
public static void main(String[] args)
{
String[] strArray1 = {"abc1", "def1", "mno1", "xyz1", "pqr1", "xyz1", "def1"};
for (int i = 0; i < strArray1.length-1; i++)
 {
for (int j = i+1; j < strArray1.length; j++)
 {
if( (strArray1[i].equals(strArray1[j])) && (i != j) )
 {
System.out.println("Duplicates : "+strArray1[j]);
}
}
}
}
}
```

Output: Duplicate Element is: def1
Duplicate Element is: xyz1

35. Can we use Generics with the array?

No, we cannot use Generic with an array.

36. How to iterate an array in java?

1) Using normal for loop
Example:

```
public class Class1
{
public static void main(String[] args)
{
int[] a1 = new int[]{46, 12, 78, 34, 89, 26};
//Iterating over an array using normal for loop
for (int i = 0; i < a1.length; i++)
{
System.out.println(a1[i]);
```

```
}
}
}
```

2) Using extended new for loop

Example:

```
public class Class1
{
public static void main(String[] args)
{
int[] a2 = new int[]{4, 12, 78, 34, 89, 2};
//Iterating over an array using extended for loop
for (int i: a2){
System.out.println(i);
}
}
}
```

37. What is the two-dimensional array?

It is an array of an array in Java.

Example:

```
int[][] a = new int[3][3];// a matrix of 3×3
```

38. Do we have 3-dimensional arrays in Java?

Yes, Java supports N-dimensional array.

Multi-dimensional array in Java is nothing but an array of arrays,

Example: a 2-dimensional array is an array of 1-dimensional array.

39. Can we make array volatile in Java?

Yes, we can make an array volatile in Java, but we only make the variable which is pointing to array volatile.

40. How to find the missing element in integer array of 1 to 7?

Solution to solve this problem is to calculate sum of all numbers in the array and compare with an expected sum, the difference would be the missing number.

int arr [] = new int[]{1,2,3,5,6,7};

Get the sum of numbers

total = n*(n+1)/2;

Subtract all the numbers from sum and you will get the missing number.

According to below logic sumOfNnumberss is 7*(7+1)/2=28

sumOfElements = 1+2+3+5+6+7=24

missing element is = 28-24=4

Code snippet:

```
int arr [] = new int[]{1,2,3,5,6,7};
int n = arr.length+1;
int total = n*(n+1)/2;
for(int i =0;i<arr.length;i++){
total -=arr[i];}
System.out.println(total);
```

41. How to find the duplicate in an array?

Code snippet:

```
String str = "HI RAJU I AM FINE RAJU"; // String with a duplicate word.
String[] words = str.split(" "); // Splitting and converting to Array .
for(int i = 0; i < words.length; i++){ //iterating array inside for loop
for(int j = i+1; j < words.length; j++){ //iterating same array inside another
for loop
if (words[i].equals(words[j])){ // Using equal method I am verifying which
word is repeating .
System.out.println( words[i]); // here it will print duplicate .
}
}
}
```

42. How to get largest and smallest number in an array?

We use two variables to store largest and smallest number.
First, we initialize largest with Integer.MIN_VALUE and
Next, we initialize smallest with Integer.MAX_VALUE.
In each iteration of the for loop, we will compare present number with largest and smallest number, and we will update
If a number is larger than largest, then it cannot be smaller than smallest.
That is it is not required check if the first condition is true,
We will use the if-else control construct, where else part will only execute if the first condition is false.

Code snippet:

```
import java.util.Arrays;
public class  MaximumMinimumArrayExample{
public static void largestAndSmallest(int[] numbers) {
int largest = Integer.MIN_VALUE;
int smallest = Integer.MAX_VALUE;
for (int number : numbers) {
if (number > largest) {
largest = number;
}
else if (number < smallest) {
smallest = number;
}
 }
System.out.println("Largest is : " + largest);
System.out.println("Smallest is : " + smallest);
 }
 }
```

43.What is the logic to reverse the array?

Declare a String Array String[] s = new String[]{“My”,”Leg”,”is”,”cut”};
Iterate it using for loop get all elements in reverse order means end point to start point.
Code Snippet:

```
class Demo
{
public static void main(String[] args)
{
String[] s = new String[]{"My","Leg","is","injured"};
for(int i=s.length-1; i>=0; i--)
{
System.out.println("reverse "+s[i]);
}
}
}
```

44. How do you find the second largest element in an array of integers?

Iterate the given array using for loop.
(if condition arr[i] > largest):
If current array value is greater than largest value then
Move the largest value to secondLargest and make current value as largest
(if condition arr[i] > secondLargest)
If the current value is smaller than largest and
greater than secondLargest then the current value becomes

code Snippet:

```
class Demo
{
public static void main(String[] args)
{
int myArr[] = { 14, 46, 47, 86, 92, 52, 48, 36, 66, 85 };
int largest = myArr[0];
int secondLargest = myArr[0];
```

```
System.out.println("The given array is:" );
for (int i = 0; i < myArr.length; i++) {
System.out.print(myArr[i]+"\t");
}
for (int i = 0; i < myArr.length; i++) {
if (myArr[i] > largest) {
secondLargest = largest;
largest = myArr[i];
} else if (myArr[i] > secondLargest) {secondLargest = myArr[i];
}
}
System.out.println("\nSecond largest number is:" + secondLargest);
}
}
```

45. Write a program to sum values of given array.

Declare and assign variable sum as sum =0.
Iterate the given array using for loop.
Add the entire array element and keep it in sum.
Code snippet:

```
class Demo
{
public static void main(String[] args)
{
int my_array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int sum = 0;
for (int i: my_array)
sum += i;
System.out.println("The sum is " + sum);
}
}
```

46. Write a program to insert an element in the specific position in the array.

Insert an element in 3rd position of the array (index->2, value->5)
Iterate the given array in reverse order using for loop.
Now insert given position and value after for loop.

```
class Demo
{
public static void main(String[] args)
 {
int[] my_array = {25, 14, 56, 15, 36, 56, 77, 18, 29, 49};
int Index_position = 2;
int newValue = 5;
System.out.println("Original Array : "+Arrays.toString(my_array));
for(int i=my_array.length-1; i > Index_position; i--){
my_array[i] = my_array[i-1];
}
my_array[Index_position] = newValue;
System.out.println("New Array: "+Arrays.toString(my_array));
}
}
```

STRINGS

47. Is String a primitive type or derived type?

String is a derived type.

48. In how many ways you can create string objects in java?

There are two ways to create string objects in java. One is using new operator and another one is using string literals. The objects created using new operator are stored in the heap memory and objects created using string literals are stored in string constant pool.

```
String s1 = new String("KODNEST"); //Creating string object using new operator
String s2 = "KODNEST";       //Creating string object using string literal
```

49. What is string constant pool?

String objects are most used data objects in Java. Hence, java has a special arrangement to store the string objects. String Constant Pool is one such arrangement. String Constant Pool is the memory space in heap Segment specially allocated to store the string objects created using string literals. In String Constant Pool, there will be no two string objects having the same content.

Whenever you create a string object using string literal, JVM first checks the content of the object to be created. If there exists an object in the string constant pool with the same content, then it returns the reference of that object. It doesn't create a new object. If the content is different from the existing objects then only it creates new object.

50. What is special about string objects as compared to objects of other derived types?

One special thing about string objects is that you can create string objects without using new operator i.e. using string literals. This is not possible with other derived types (except wrapper classes). One more special thing about strings is that you can concatenate two string objects using '+'. This is the relaxation java gives to string objects as they will be used most of

the time while coding. And also java provides string constant pool to store the string objects.

51. What do you mean by mutable and immutable objects?

Immutable objects are like constants. You can't modify them once they are created. They are final in nature. Whereas mutable objects are concerned, you can perform modifications to them.

52. What is the difference between String, StringBuffer and StringBuilder?

String objects created using java.lang.String class are immutable. Once they are created, they cannot be modified. If you try to modify them, a new string object will be created with modified content. This property of String class may cause some memory issues for applications which need frequent modification of string objects. To overcome this behaviour of String class, two more classes are introduced in Java to represent the strings. They are StringBuffer and StringBuilder. Both these classes are also members of java.lang package same as String class.

53. Why StringBuffer and StringBuilder classes are introduced in java when there already exists String class to represent the set of characters?

The objects of String class are immutable in nature i.e. you can't modify them once they are created. If you try to modify them, a new object will be created with modified content. This may cause memory and performance issues if you are performing lots of string modifications in your code. To overcome these issues, StingBuffer and StringBuilder classes are introduced in java.

54. How many objects will be created in the following code and where they will be stored in the memory?

```
String s1 = "KODNEST";
String s2 = "KODNEST";
```

Only one object will be created and this object will be stored in the string constant pool.

55. How do you create mutable string objects?

Using StringBuffer and StringBuilder classes. These classes provide mutable string objects.

56. Which one will you prefer among "==" and equals() method to compare two string objects?

I prefer equals() method because it compares two string objects based on their content. That provides more logical comparison of two string objects. If you use "==" operator, it checks only references of two objects are equal or not. It may not be suitable in all situations. So, rather stick to equals() method to compare two string objects.

Note: "==" operator compares the two objects on their physical address. That means if two references are pointing to same object in the memory, then comparing those two references using "==" operator will return true. For example, if s1 and s2 are two references pointing to same object in the memory, then invoking s1 == s2 will return true. This type of comparison is called "Shallow Comparison".

equals() method, if not overridden, will perform same comparison as "==" operator does i.e. comparing the objects on their physical address. So, it is always recommended that you should override equals() method in your class so that it provides field by field comparison of two objects. This type of comparison is called "Deep Comparison".

In java.lang.String class, equals() method is overridden to provide the comparison of two string objects based on their contents. That means, any two string objects having same content will be equal according to

equals() method. For example, if s1 and s2 are two string objects having the same content, then invoking s1.equals(s2) will return true.

hashCode() method returns hash code value of an object in the Integer form. It is recommended that whenever you override equals() method, you should also override hashCode() method so that two equal objects according to equals() method must return same hash code values. This is the general contract between equals() and hashCode() methods that must be maintained all the time.

In java.lang.String class, hashCode() method is also overridden so that two equal string objects according to equals() method will return same hash code values. That means, if s1 and s2 are two equal string objects according to equals() method, then invoking s1.hashCode() == s2.hashCode() will return true.

Let's apply these three methods on string objects and try to analyse their output.

Define two string objects like below,
String s1 = "KODNEST";
String s2 = "KODNEST";
Now apply above methods on these two objects.
s1 == s2 —> will return true as both are pointing to same object in the constant pool.
s1.equals(s2) —> will also return true as both are referring to same object.
s1.hashCode() == s2.hashCode() —> It also returns true.

57. How do you convert given string to char array?

Using toCharArray() method.

58. How many objects will be created in the following code and where they will be stored?

String s1 = new String("abc");

String s2 = "abc";
Here, two string objects will be created. Object created using new operator(s1) will be stored in the heap memory. The object created using string literal(s2) is stored in the string constant pool.

59. What is string intern?

String object in the string constant pool is called as String Intern. You can create an exact copy of heap memory string object in string constant pool. This process of creating an exact copy of heap memory string object in the string constant pool is called interning. intern() method is used for interning

60. How many objects will be created in the following code and where they will be stored?

String s1 = new String("KODNEST");
String s2 = new String("KODNEST");
Two objects will be created and they will be stored in the heap memory.

61. Can we call String class methods using string literals?

Yes, we can call String class methods using string literals. Here are some examples,
"KODNEST".charAt(0) ;
"KODNEST".compareTo("KODNEST") ;
"KODNEST".indexOf('O');

62. How do you remove all white spaces from a string in java?

1) Using replaceAll() Method.

In the first method, we use replaceAll() method of String class to remove all white spaces (including tab also) from a string. This is the one of the easiest method to remove all white spaces from a string. This method takes two parameters. One is the string to be replaced and another one is the string to be replaced with. We pass the string "\s" to be replaced with an empty string "".

2) Without Using replaceAll() Method.

In the second method, we remove all white spaces (including tab also) from a string without using replaceAll() method. First we convert the given string to char array and then we traverse this array to find white spaces. We append the characters which are not the white spaces to StringBuffer object.

```
class RemoveWhiteSpaces
{
   public static void main(String[] args)
   {
      String str = "  Core Java jsp servlets           jdbc struts hibernate spring
";

      //1. Using replaceAll() Method

      String strWithoutSpace = str.replaceAll("\\s", "");

      System.out.println(strWithoutSpace);        //Output :
CoreJavajspservletsjdbcstrutshibernatespring

      //2. Without Using replaceAll() Method

      char[] strArray = str.toCharArray();

      StringBuffer sb = new StringBuffer();

      for (int i = 0; i < strArray.length; i++)
      {
         if( (strArray[i] != ' ') && (strArray[i] != '\t') )
         {
            sb.append(strArray[i]);
```

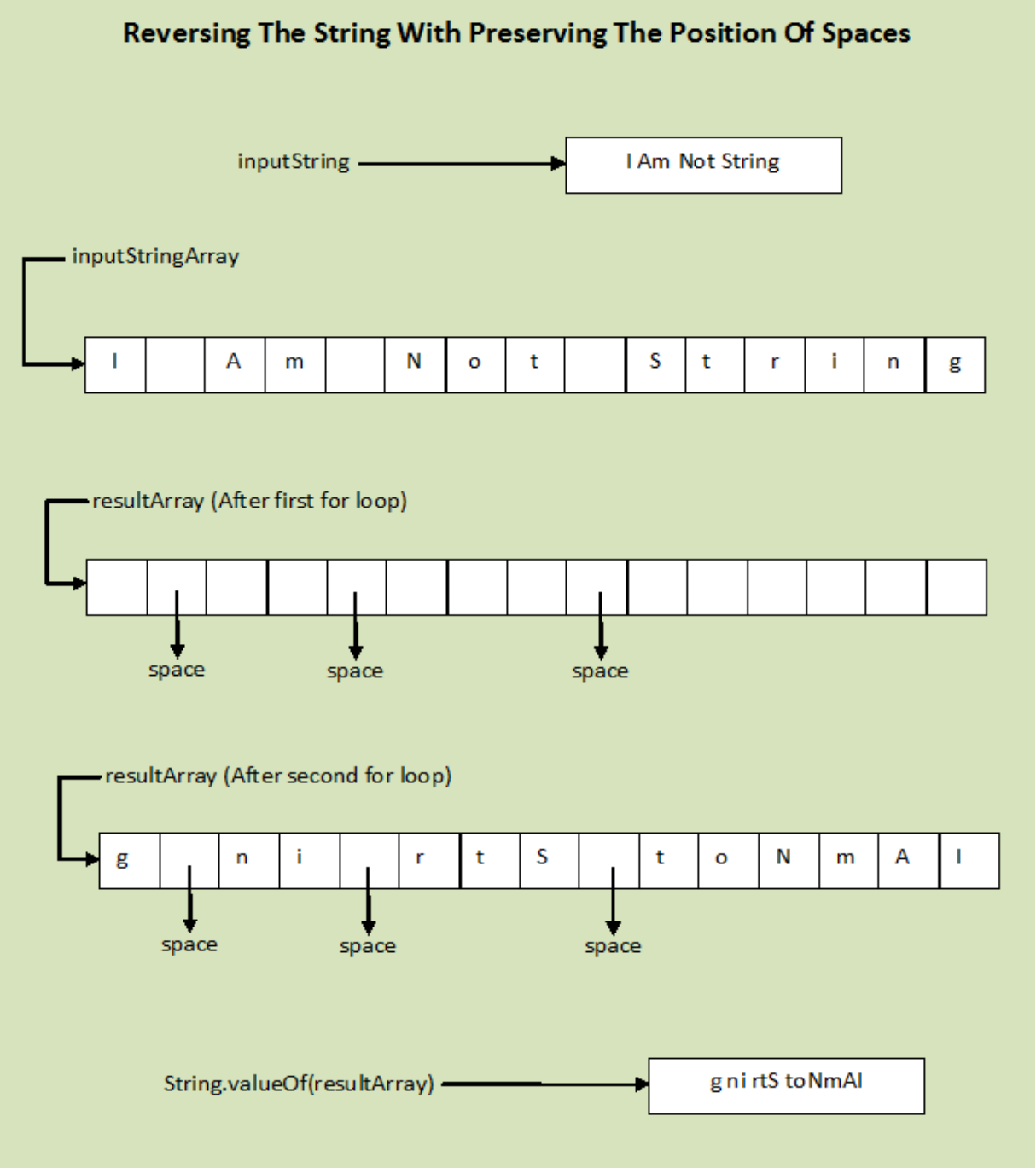
```
        }
      }

      System.out.println(sb);          //Output :
CoreJavajspservletsjdbcstrutshibernatespring
   }
}
```

63. Write a java program to reverse a given string with preserving the position of spaces?

Write a java program to reverse a string with preserving the position of spaces. For example, if "I Am Not String" is the given string then the reverse of this string with preserving the position of spaces is "g ni rtS toNmAI". Notice that the position of spaces in the original string are maintained in the reversed string also. That means, if the given string has a space at index 3, then there should be also a space in the reversed string at index 3.

First, we convert the given 'inputstring' to char array and call it as 'inputStringArray'. We define one more char array called 'resultArray' with the same size as 'inputStringArray'. In the first for loop, for every space in the 'inputStringArray', we insert space in the 'resultArray' at the corresponding positions. In the second for loop, we copy non-space characters of 'inputStringArray' starting from first to last into the 'resultArray' at 'j' position where 'j' will have value from length of resultArray to . Before copying, we check whether the 'resultArray' already contains a space at index 'j' or not. If it contains, we copy the character in the next position. See the below image for more clarity.




```
public class MainClass1
{
   static void reverseString(String inputString)
   {
      //Converting inputString to char array 'inputStringArray'

      char[] inputStringArray = inputString.toCharArray();
```

```
    //Defining a new char array 'resultArray' with same size as
inputStringArray

    char[] resultArray = new char[inputStringArray.length];

    //First for loop :
    //For every space in the 'inputStringArray',
    //we insert spaces in the 'resultArray' at the corresponding positions

    for (int i = 0; i < inputStringArray.length; i++)
    {
       if (inputStringArray[i] == ' ')
       {
          resultArray[i] = ' ';
       }
    }

    //Initializing 'j' with length of resultArray

    int j = resultArray.length-1;

    //Second for loop :
    //we copy every non-space character of inputStringArray
    //from first to last at 'j' position of resultArray

    for (int i = 0; i < inputStringArray.length; i++)
    {
       if (inputStringArray[i] != ' ')
       {
          //If resultArray already has space at index j then decrementing 'j'

          if(resultArray[j] == ' ')
          {
             j--;
```

```
            }

            resultArray[j] = inputStringArray[i];

            j--;
          }
       }

       System.out.println(inputString+" ---> "+String.valueOf(resultArray));
    }

    public static void main(String[] args)
    {
       reverseString("I Am Not String");

       reverseString("JAVA JSP ANDROID");

     }
}
```

Output:

I Am Not String —> g ni rtS toNmAI
JAVA JSP ANDROID —> DIOR DNA PSJAVAJ

64. Is String Thread Safe In Java?

Yes string is thread safe in Java as String is immutable.

65. What Is StringBuffer In Java?

StringBuffer class is the companion class of String. StringBuffer is a mutable(modifiable) sequence of characters which is in contrast to String class which is an immutable sequence of characters. Thus in case of StringBuffer length and content of the sequence can be changed through certain method calls.

Since StringBuffer is mutable a new String object is not created every time string is modified, which in turn results in less memory consumptions and not having lots of intermediate String object for garbage collection.

66. What Is StringBuilder In Java?

StringBuilder class (Added in Java 5), just like StringBuffer, is a mutable (modifiable) sequence of characters which is in contrast to String class which is an immutable sequence of characters. Thus in case of StringBuilder length and content of the sequence can be changed through certain method calls.

67. How to Get Characters And Substrings By Index With In A String?

You can get the character at a particular index within a string by invoking the charAt() accessor method.
String str = "Example String";
char resChar = str.charAt(3);
Will give char 'm'. If you want to get more than one consecutive character from a string, you can use the substring method. The substring method has two versions -
String subString(int beginIndex, int endIndex) - Returns a new string that is a substring of this string.
String subString(int beginIndex) - Returns a new string that is a substring of this string.

68. How Can You Find Characters Or Substrings Within A String?

To find characters or substrings with in a string indexOf() and lastIndexOf() methods can be used.
You can also use contains() method
public boolean contains(CharSequence s) - Returns true if and only if this string contains the specified sequence of char values. Otherwise it returns false.

69. How Can You Split A String In Java?

String provides a split method in order to split the string into one or more substring based on the given regular expression.

As example: If you have a string where one (or more) spaces are used and you want to split it around those spaces.

```
String str1 = "split example    program";
String[] strArray = str1.split("\s+");
```

70. How Can You Join Strings In Java?

With Java 8 join() method has been added in the String class which makes it very easy to join the multiple strings.
join method has two overloaded versions -
public static String join(CharSequence delimiter, CharSequence... elements) - Returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) – Here elements is an Iterable that will have its elements joined together and delimiter is a sequence of characters that is used to separate each of the elements in the resulting String.

71. How to convert String to Date in Java?

Prior to Java 8, you can use DateFormat or SimpleDateFormat class to convert a String to Date In Java or vice-versa. From Java 8 onwards, when you use the new Date and Time API, you can also use the DateTimeFormatter class to convert String to LocalDate, LocalTime, or LocalDateTime class in Java.

72. Difference between format() and printf() method in Java?

Even though both methods can be used to format String and they have same rules the key difference is that format() method returns a formatted String while printf() method print formatted String to console. So, if you

need a formatted String, use format method and if you want to print, then use the printf() method.

73. How To Convert String To Integer In Java?

There are two methods available in java to convert string to integer. One is Integer.parseInt() method and another one is Integer.valueOf() method. Both these methods are static methods of java.lang.Integer class. Both these methods throw NumberFormatException if input string is not a valid integer. The main difference between Integer.parseInt() and Integer.valueOf() method is that parseInt() method returns primitive int where as valueOf() method returns java.lang.Integer object.

Example1:

```
public class Demo1
{
   public static void main(String[] args)
   {
      String s = "2015";

      int i = Integer.parseInt(s);

      System.out.println(i);        //Output : 2015
   }
}
```

Example2:

```
public class Demo2
{
   public static void main(String[] args)
   {
      String s = "2015";

      int i = Integer.valueOf(s);

      System.out.println(i);        //Output : 2015
```

```
    }
}
```

74. How to Convert Integer to String in Java?

You are also often needed to do the reverse conversion i.e. converting from integer to string. Java provides couple of methods to do that also. One is Integer.toString() method and another one is String.valueOf() method. Both these methods return string representation of the given integer.

Example1:

```
public class Demo1
{
   public static void main(String[] args)
   {
      int i = 2015;

      String s = Integer.toString(i);

      System.out.println(s);     //Output : 2015
   }
}
```

Example2:

```
public class Demo2
{
   public static void main(String[] args)
   {
      int i = 2015;

      String s = String.valueOf(i);

      System.out.println(s);     //Output : 2015
   }
}
```

75. Write a Java program to swap first and last characters of words in a sentence.

Input : kodnest for freshers
Output :todnesk rof sresherf

Approach: As mentioned in the example we have to replace first and last character of word and keep rest of the alphabets as it is.
First we will create an Char array of given String by using toCharArray() method.
Now we iterate the char array by using for loop.
In for loop, we declare a variable whose value is dependent on i.
Whenever we found an alphabet we increase the value of i and whenever we reach at space, we are going to perform swapping between first and last character of the word which is previous of space.

Code Snippet:

```
class Demo
{
   static String count(String str)
   {
      // Create an equivalent char array
      // of given string
      char[] ch = str.toCharArray();
      for (int i = 0; i < ch.length; i++) {

         // k stores index of first character
         // and i is going to store index of last
         // character.
         int k = i;
         while (i < ch.length && ch[i] != ' ')
            i++;
```

```
            // Swapping
            char temp = ch[k];
            ch[k] = ch[i - 1];
            ch[i - 1] = temp;

            // We assume that there is only one space
            // between two words.
        }
        return new String(ch);
    }

    public static void main(String[] args)
    {
        String str = "kodnest for freshers";
        System.out.println(count(str));
    }
}
```

METHODS

76. What is method overloading?

When a class has more than one method with same name but different parameters, then we call those methods are overloaded. Overloaded methods will have same name but different number of arguments or different types of arguments.

77. What is method signature? What are the things it consists of?

Method signature is used by the compiler to differentiate the methods. Method signature consists of three things.

a) Method name

b) Number of arguments

c) Types of arguments

78. Can we declare one overloaded method as static and another one as non-static?

Yes. Overloaded methods can be either static or non static.

79. How does compiler differentiate overloaded methods from duplicate methods?

Compiler uses method signature to check whether the method is overloaded or duplicated. Duplicate methods will have same method signatures i.e. same name, same number of arguments and same types of arguments. Overloaded methods will also have same name but differ in number of arguments or else types of arguments

80. Is it possible to have two methods in a class with same method signature but different return types?

No, compiler will give duplicate method error. Compiler checks only method signature for duplication not the return types. If two methods have same method signature, straight away it gives compile time error.

81. In "MyClass" , there is a method called "myMethod" with four different overloaded forms. All four different forms have different visibility (private, protected, public and default). Is "myMethod" properly overloaded?

Yes. Compiler checks only method signature for overloading of methods not the visibility of methods.

82. Can overloaded methods be synchronized?

Yes. Overloaded methods can be synchronized.

83. Can we overload main() method?

Yes, we can overload main() method. A class can have any number of main() methods but execution starts from public static void main(String[] args) only.

84. Can we declare overloaded methods as final?

Yes, we can declare overloaded methods as final

85. In the below class, is constructor overloaded or is method overloaded?

```
public class A
{
   public A()
   {
      //-----> (1)
   }

   void A()
   {
      //-----> (2)
   }
}
```

None of them. It is neither constructor overloaded nor method overloaded. First one is a constructor and second one is a method.

86. Overloading is the best example of dynamic binding. True or false?

False. Overloading is the best example for static binding.

87. Can overloaded method be overridden?

Yes, we can override a method which is overloaded in super class

88. Is overloading is a run-time or compile-time polymorphism?

Compile-time polymorphism, as it is resolved at compile-time

89. Is method overloading is a static binding or dynamic binding?

Static binding, as it is resolved during compile-time

90. What are the other names used to refer method overloading?

Compile-time polymorphism or Static binding

91. What are the restrictions on access modifier in method signature while overloading in Java?

Access modifiers doesn't affect method overloading, so overloaded methods can have same or different access levels

92. Why it is not possible to overload methods based on the return type?

Reason is type ambiguity

First of all, we cannot have two same methods with exactly same input parameters. In this case, compiler throws error

It is a compile-time error, as it is resolved during compile-time

Also, it is very difficult for JVM to understand as to which version of overloaded methods to call

93. Why method overloading required in Java?

Suppose, if we want perform similar kind of tasks and their operation differ only by number of parameters or their data-types or both then method overloading is the best concept to apply

Maintains consistency with method naming for similar type of tasks increases the readability of the program

This helps the developer to invoke method with same name but changing required arguments in required order with their corresponding data-types

Example: java.lang.String class from java.lang package contains 9 overloaded 'valueOf()' method with different number of input parameters or their data-types

CONSTRUCTOR

94. Define Constructor?

Constructor is a special method given in OOP language for creating and initializing object.
In java, constructor role is only initializing object, and new keyword role is crating object.

95. What Are the Rules associated In Defining a Constructor?

1. Constructor name should be same as class name.
2. It should not contain return type.
3. It should not contain Non Access Modifiers:
final, static, abstract, synchronized
In it logic return statement with value is not allowed.
4. It can have all four accessibility modifiers:
private, public, protected, default
5. It can have parameters
6. It can have throws clause:
7. We can throw exception from constructor.
8. It can have logic, as part of logic it can have all java legal statement except return statement with value.
9. We cannot place return in constructor.

96. Can We Define A Method With Same Name Of Class?

Yes, it is allowed to define a method with same class name. No compile time error and no runtime error are raised, but it is not recommended as per coding standards.

97. If We Place Return Type In Constructor Prototype Will It Leads To Error?

No, because compiler and JVM considers it as a method.

98. How Compiler And Jvm Can Differentiate Constructor And Method Definitions Of Both Have Same Class Name?

By using return type, if there is a return type it is considered as a method else it is considered as constructor.

99. How Compiler And Jvm Can Differentiate Constructor And Method Invocations Of Both Have Same Class Name?

By using new keyword, if new keyword is used in calling then constructor is executed else method is executed.

100. Why Return Type Is Not Allowed For Constructor?

As there is a possibility to define a method with same class name, return type is not allowed to constructor to differentiate constructor block from method block.

101. Why Constructor Name Is Same As Class Name?

Every class object is created using the same new keyword, so it must have information about the class to which it must create object.
For this reason constructor name should be same as class name.

102. Can We Declare Constructor As Private?

a. Yes we can declare constructor as private.
b. All four access modifiers are allowed to constructor.
c. We should declare constructor as private for not to allow user to create object from outside of our class.
d. Basically we will declare private constructor in Singleton design pattern.

103. Is Constructor Definition Is Mandatory In Class?

No, it is optional. If we do not define a constructor compiler will define a default constructor.

104. Why Compiler Given Constructor Is Called As Default Constructor?

Because it obtains all its default properties from its class.
They are:

1. Its accessibility modifier is same as its class accessibility modifier
2. Its name is same as class name
3. It does not have parameters and logic

105. What Is Default Accessibility Modifier Of Default Constructor?

It is assigned from its class.

106. When Compiler Provides Default Constructor?

Only if there is no explicit constructor defined by developer.

107. When Developer Must Provide Constructor Explicitly?

If we want do execute some logic at the time of object creation, that logic may be object initialization logic or some other useful logic.

108. If Class Has Explicit Constructor, Will It Has Default Constructor?

No. compiler places default constructor only if there is no explicit constructor.

109. What is No-arg constructor?

Constructor without any arguments is called no-arg constructor. Default constructor in java is always known as no-arg constructor.

```
class MyClass

{

public MyClass()

{

//No-arg constructor

}
```

}

110. What is constructor chaining and how can it be achieved in Java

Constructor chaining is the process of calling one constructor from another constructor with respect to current object.

Constructor chaining can be done in two ways:

1. Within same class: It can be done using this() keyword for constructors in same class

2. From base class: by using super() keyword to call constructor from the base class.

Constructor chaining occurs through inheritance. A sub class constructor's task is to call super class's constructor first. This ensures that creation of sub class's object starts with the initialization of the data members of the super class. There could be any numbers of classes in inheritance chain. Every constructor calls up the chain till class at the top is reached.

111. Why do we need constructor chaining?

This process is used when we want to perform multiple tasks in a single constructor rather than creating a code for each task in a single constructor we create a separate constructor for each task and make their chain which makes the program more readable.

112. Can we use this() and super() in a method?

No, we can't use this() and super() in a method.

```
class SuperClass
{
public SuperClass()
{
System.out.println("Super Class Constructor");
}
}
class SubClass extends SuperClass
{
public SubClass()
```

```
{
System.out.println("Sub Class Constructor");

}
void method()
{
this(); //Compile time error
super(); //Compile time error
}
}
```

113. Does a constructor create the object?

'New' operator in Java creates the objects. Constructor comes in the later stage in object creation. Constructor's job is to initialize the members after the object has reserved memory for itself.

114. What are the common uses of "this" keyword in java?

"this" keyword is a reference to the current object and can be used for the following -

1. Passing itself to another method.
2. Referring to the instance variable when local variable has the same name.
3. Calling another constructor in constructor chaining.

115. Can we call sub class constructor from super class constructor?

No. There is no way in java to call sub class constructor from a super class constructor.

116. What happens if you keep return type for a constructor?

It will be treated as a normal method. But compiler gives a warning saying that method has a constructor name.

```
class MyClass
{
```

```
    int MyClass()
    {
        return 0;    //No Compile time error but just a warning
    }
}
```

ENCAPSULATION

117. What is Encapsulation?

It is the technique of making the fields in a class private and providing access to these fields with the help of public methods. If a field is declared private, it cannot be accessed by anyone outside the class, thereby hiding the fields within the class. Therefore encapsulation is also referred to as data hiding.

118. What is the primary benefit of encapsulation?

The main benefit of encapsulation is the ability to modify the implemented code without breaking the code of others who use our code. It also provides us with maintainability, flexibility and extensibility to our code.

119. What is the difference between encapsulation and abstraction?

1. Abstraction solves the problem at design level while encapsulation solves the problem at implementation level.
2. Abstraction is used for hiding the unwanted data and provides only the required data. On the other hand encapsulation means hiding the code and data into a single unit to protect the data from outside world.
3. Abstraction lets you focus on what the object does instead of how it does it while Encapsulation means hiding the internal details or mechanics of how an object does something.
4.For example: Outer Look of a Television i.e. it has a display screen and channel buttons to change channel explains 'abstraction' but inner implementation detail of a television i.e. how CRT and display screen are connected with each other using different circuits explains 'encapsulation'.

120. What are the features of encapsulation?

Encapsulation means combining the data of our application and its manipulation at one place. Encapsulation allows the state of an object to be accessed and modified through behaviour. It reduces the coupling of modules and increases the cohesion inside them.

121. Explain in detail encapsulation in Java?

Encapsulation is nothing but protecting anything which is prone to change. Rational behind encapsulation is that if any functionality which is well encapsulated in code i.e. maintained in just one place and not scattered around code is easy to change. This can be better explained with a simple example of encapsulation in Java. We all know that constructor is used to create object in Java and constructor can accept argument. Suppose we have a class 'Loan' which has a constructor and in various classes we have created instance of 'loan' using this constructor. Now requirements will change and you need to include 'age of borrower' as well while taking loan. Since this code is not well encapsulated i.e. not confined in one place you need to change at every place where you are calling this constructor i.e. for one change you need to modify several files instead of just one file which is more error prone and tedious. Though it can be done with refactoring feature of advanced IDE it would prove better if we only need to make change at one place. This is possible if we encapsulate 'Loan' creation logic in one method say 'createLoan()'. The code written for client will call this method and this method internally creates 'Loan' object. In this case you only need to modify this method instead of the whole client code.

122. Give an example of Encapsulation in Java.

```
class Loan
{
private int duration; //private variables examples of encapsulation
private String loan;
private String borrower;
private String salary;
```

```
//public constructor can break encapsulation instead use factory method
private Loan(int duration, String loan, String borrower, String salary)
{
this.duration = duration;
this.loan = loan;
this.borrower = borrower;
this.salary = salary;
}

// create loan can encapsulate loan creation logic
public Loan createLoan(String loanType)
{
return loan;
}
}
```

123. What are the advantages of using encapsulation in Java and OOPS?

Below mentioned are few advantages of using encapsulation while writing code in Java or any Object oriented programming language:

1. Encapsulated Code is more flexible and easy to change with inception of new requirements.
2. Encapsulation in Java makes unit testing easy.
3. Encapsulation in Java allows you to control who can access what.
4. Encapsulation also helps to write immutable class in Java which is a good choice in multi-threading environment.
5. Encapsulation reduces coupling of modules and increases cohesion inside a module because all the pieces of one thing are encapsulated in one place.
6. Encapsulation allows you to change one part of code without affecting other part of code.

124. What should you encapsulate in code?

Anything which can be changed or which is more likely to be changed in near future is candidate of encapsulation. This also helps to write more specific and cohesive code. For instance object creation code, code which can be improved in future like sorting and searching logic.

125. Mention some important points about encapsulation in Java.

1. "Whatever changes encapsulate it" is a famous design principle.
2. Encapsulation helps in loose coupling and high cohesion of code.
3. Encapsulation in Java is achieved using access modifiers private, protected and public.
4. 'Factory pattern' and 'singleton pattern' in Java makes good use of encapsulation

126. Give an example of how to achieve encapsulation in Java?

To achieve encapsulation in Java:

- Declare the variables of a class as 'private'.
- Provide public setter and getter methods to modify and view the variable's values

```
public class EncapTest
{
private String name;
private String idNum;
private int age;
public int getAge()
{
return age;
}
public String getName()
{
return name;
}
public String getIdNum()
```

```
{
return idNum;
}
public void setAge( int newAge)
{
age = newAge;
}
public void setName(String newName)
{
name = newName;
}

public void setIdNum( String newId)
{
idNum = newId;
}
}
```

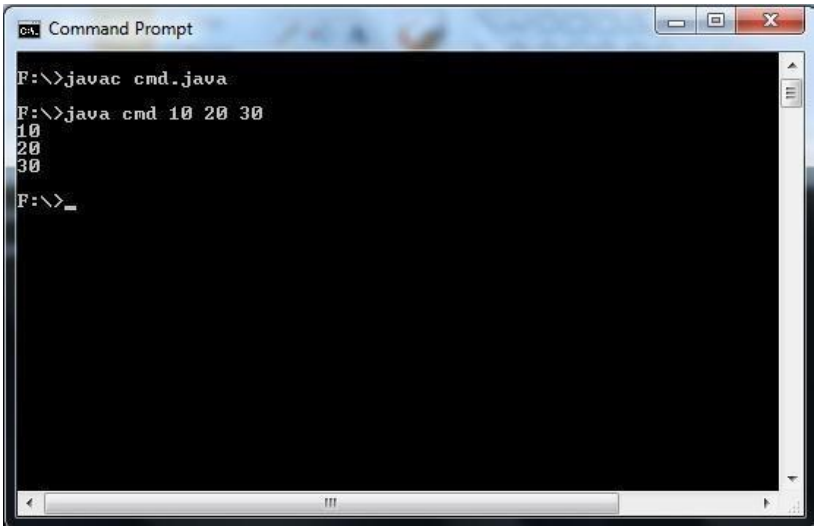
COMMAND LINE ARGUMENTS

127. What are Command line arguments in Java?

The command line argument is the argument passed to a program at the time when you run it. To access the command-line argument inside a java program is quite easy, they are stored as string in String array passed to the args parameter of main() method.

Example:

```
class cmd
{
  public static void main(String[] args)
  {
    for(int i=o;i< args.length;i++)
    {
    System.out.println(args[i]);
    }
  }
}
```



STATIC

128. What is static in java?

Static is a Non Access Modifier.
Static can be applied to variable, method, nested class and initialization blocks (static block).

129. What is a static variable?

A Static variable gets memory allocated only once during the time of class loading.
All the instance of the class share the same copy of the variable, a static variable can be accessed directly by calling "<<ClassName>>.<<VariableName>>" without need to create instance for the class.
value of a static variable will be common for all instances
example:

```
public class StaticVariableExample
{
   static int a =10;
   public static void main(String args[]){
      StaticVariableExample s1 = new StaticVariableExample();
      StaticVariableExample s2 = new StaticVariableExample();
      System.out.println("s1.a value :"+s1.a);
      System.out.println("s2.a value :"+s2.a);
      //Change s1 a value alone
      s1.a=20;
      System.out.println("s1.a value :"+s1.a);
      System.out.println("s2.a value :"+s2.a);
   }
}
```

Output will be
s1.a value :10

s2.a value :10
s1.a value :20
s2.a value :20

note: Local variables cannot be assigned as static it will throw compile time error “illegal start of expression”, as the memory cannot be assigned during class load.

130. What is a static method ?

A static method belongs to class rather than object. It can be called directly by using the class name
“<<ClassName>>.<<MethodName>>”
A static method can access static variables directly and it cannot access non-static variables and can only call a static method directly and it cannot call a non-static method from it.
Only the main() method which is static will be called by the JVM automatically, Not all the static method will be called automatically.

131. Can a static block exist without a main() method ?

Yes. You can have static block alone in the class without a main method.

132. Can we overload static methods in Java?

Yes, you can overload a static method in Java.

133. Can we override static methods in Java?

No, you cannot override a static method in Java as there will not be any Run-time Polymorphism happening.

134. Why main() method is declared as static ?

If our main() method is not declared as static then the JVM has to create object first and call which causes the problem of having extra memory allocation.

135. What is a static block?

A static block is a block of code inside a Java class that will be executed when a class is first loaded in to the JVM. Mostly the static block will be used for initializing the variables.

Static block will be called only one while loading and it cannot have any return type, or any keywords (this or super).

```
class test
{
        int val;
         static{
     val = 100;
   }
}
```

136. Can we have multiple static blocks in our code?

Yes, we can have more than one static block in our code. It will be executed in the same order it is written.

137. What is a static class?

In Java only nested classes are allowed to be declared as static, a top level class cannot be declared as static.

Even though static classes are nested inside a class, they don't need the reference of the outer class they act like outer class only.

138. Can constructors be static in Java?

In general a static method means that "The Method belong to class and not to any particular object" but a constructor is always invoked with respect to an object, so it makes no sense for a constructor to be static.

139. Why abstract method cannot be static in Java?

Suppose when you have a concrete method in an abstract class then that method can be static. Suppose we have a class like below

```
public class AbstractTest
{
   static void disp()
```

```
    {
        System.out.println("disp of static method");
    }
}
```

Then the disp() can be access by “AbstractTest.disp()”

However, for the same reason cannot be applied when you declare a static method to be abstract. Since static method can be called directly, making it abstract would make it possible to call an undefined method which is of no use, hence it is not allowed.

140. Can Interface in Java have static methods in it?

No, Interface cannot have static methods in it because all methods are implicitly abstract. This is why an interface cannot have a static method.

141. Can abstract class have static variable in it?

Yes, an abstract class can have static variables in it.

142. Non-static method cannot be referenced from a static context?

```
public class Test
{

    public static void main(String[] args)
    {
        welcome();
    }

    void welcome()
    {
        System.out.println("Welcom to JavaInterviewPoint");
    }
}
```

The welcome() method which we tried calling is an instance-level method, we do not have an instance to call it . Static methods belong to the class, non-static methods belong to instances of the class and hence it

throws the error ”non-static method cannot be referenced from a static context “.

143. What is the output of this Program?

```
class Test1
{
public static void main(String[] args)
   {
      int x = 20;
      System.out.println(x);
   }
   static
   {
      int x = 10;
      System.out.print(x + " ");
   }
}
```

Answer: 10 20

144. What is the output of this Program?

```
class Test1
{
   int x = 10;
public  static void main(String[] args)
   {
      System.out.println(x);
   }
   static
   {
      System.out.print(x + " ");
   }
}
```

Answer: Error

Because , If we are trying to print the instance variable inside the static block or static method without creating class instance then it will give the error : non-static variable x cannot be referenced from a static context.

145. What is the output of this Program?

```
class Test1 {
   int x = 10;
public
   static void main(String[] args)
   {
      Test1 t1 = new Test1();
      System.out.println(t1.x);
   }
   static
   {
      int x = 20;
      System.out.print(x + " ");
   }
}
```

Answer: 20 10

Because we can print the instance variable inside the static method after creating the class reference.

146. What is the output of this Program?

```
class Test1 {
   int x = 10;
public
   static void main(String[] args)
   {
      System.out.println(Test1.x);
   }
   static
   {
      int x = 20;
```

```
        System.out.print(x + " ");
    }
}
```

Answer: Error

Because we cannot access the instance variable with class name.

Otherwise it will give the error: non-static variable x cannot be referenced from a static context

147. What is the output of this Program?

```
class Test1 {
    static int x = 10;
public
    static void main(String[] args)
    {
        Test1 t1 = new Test1();
        Test1 t2 = new Test1();

        t1.x = 20;
        System.out.print(t1.x + " ");
        System.out.println(t2.x);
    }
}
```

Answer: 20 20

Because static variable is class level variable. If we do update in any reference then automatically all pointing reference values are changed.

148. What is the output of this program?

```
class Test1 {
    static int i = 1;
public static void main(String[] args)
    {
        for (int i = 1; i < 10; i++) {
            i = i + 2;
            System.out.print(i + " ");
```

```
    }
  }
}
```

Answer: 3 6 9

Here local variables are printed after execution. If we want to execute static variables, then we write Test1.i or we write Test1 object.i.

149. What is the output of this question?

```
class Test1
{
  static int i = 1;
public static void main(String[] args)
  {
    int i = 1;
    for (Test1.i = 1; Test1.i < 10; Test1.i++) {
      i = i + 2;
      System.out.print(i + " ");
    }
  }
}
```

Answer: 3 5 7 9 11 13 15 17 19

Because, Here, two different copies of variable are declared, one is static and other one is local. If we write Test1.i then, static variable is executed and if we write only i, then local variable are executed.

150. What is the output of this question?

```
class Test1 {
  static int i = 1;
public static void main(String[] args)
  {
    static int i = 1;
    for (Test1.i = 1; Test1.i < 10; Test1.i++) {
      i = i + 2;
      System.out.print(i + " ");
```

```
    }
  }
}
```

Answer: Error

Because we cannot declare the static variable inside the block. If we declare static variable inside the block, then we will get the compile time error: illegal start of expression.

151. What is the output of this question?

```
class Test1
 {
public static void main(String[] args)
  {
    static int arr1[] = { 11, 22, 33 };
    static int arr2[] = { 11, 22, 33, 44, 55 };
    static int ptr[];
    ptr = arr1;
    arr1 = arr2;
    arr2 = ptr;
    System.out.print(arr1.length + " ");
    System.out.println(arr2.length);
  }
}
```

Answer: Error

Because, here we are trying to declare array as static type but we cannot declare the local array as static type. If we will try to declare the local variable as static, then we will get error: illegal start of expression.

INHERITANCE

152. What is Inheritance in Java?

Inheritance is an Object oriented feature which allows a class to inherit behaviour and data from other class.

For example, a class Car can extend basic feature of Vehicle class by using Inheritance.

One of the most intuitive examples of Inheritance in the real world is Father-Son relationship, where Son inherits Father's property.

153. What are different types of Inheritance supported by Java?

Java supports single Inheritance, multi-level inheritance and at some extent multiple inheritances because Java allows a class to only extend another class, but an interface in Java can extend multiple inheritances.

154. Why Inheritance is used by Java Programmers?

Inheritance is used for code reuse and leveraging Polymorphism by creating a type hierarchy.

It's better to use Inheritance for type declaration but for code reuse composition is a better option because it's more flexible.

155. How to use Inheritance in Java?

You can use Inheritance in Java by extending classes and implementing interfaces.

Java provides two keywords extends and implements to achieve inheritance.

A class which is derived from another class is known as a subclass and an interface which is derived from another interface is called sub-interface.

A class which implements an interface is known as implementation.

156. What is the syntax of Inheritance?

You can use either extends of implements keyword to implement Inheritance in Java. A class extends another class using extends keyword, an interface can extend another interface using extend keyword, and a class can implement an interface using implements keyword in Java.

157. What is the difference between Inheritance and Encapsulation?

Inheritance is an object oriented concept which creates a parent-child relationship.

It is one of the ways to reuse the code written for parent class but it also forms the basis of Polymorphism.

On the other hand, Encapsulation is an object oriented concept which is used to hide the internal details of a class e.g. HashMap encapsulate how to store elements and how to calculate hash values.

158. What is the difference between Inheritance and Abstraction?

Abstraction is an object oriented concept which is used to simply things by abstracting details. It helps in the designing system.

On the other hand, Inheritance allows code reuse. You can reuse the functionality you have already coded by using Inheritance.

159. What is the difference between Polymorphism and Inheritance?

Both Polymorphism and Inheritance goes hand on hand, they help each other to achieve their goal. Polymorphism allows flexibility; you can choose which code to run at runtime by overriding.

160 What is the difference between Composition and Inheritance in OOP?

1. The Composition is more flexible because you can change the implementation at runtime by calling setXXX() method, but Inheritance cannot be changed i.e. you cannot ask a class to implement another class at runtime.

2. Composition builds HAS-A relationship while Inheritance builds IS-A relationship e.g. A Room HAS A Fan, but Mango IS-A Fruit.

3. The parent-child relationship is best represented using Inheritance but If you just want to use the services of another class use Composition.

161. Can we override static method in Java?

No, you cannot override a static method in Java because it's resolved at compile time.
In order for overriding to work, a method should be virtual and resolved at runtime because objects are only available at runtime.

162. Can we overload a static method in Java?

Yes, you can overload a static method in Java. Overloading has nothing to do with runtime but the signature of each method must be different. In Java, to change the method signature, you must change either number of arguments, type of arguments or order of arguments.

163. Can we override a private method in Java?

No, you cannot override a private method in Java because the private method is not inherited by the subclass in Java, which is essential for overriding. In fact, a private method is not visible to anyone outside the class and, more importantly, a call to the private method is resolved at compile time by using Type information as opposed to runtime by using the actual object.

164. What is method hiding in Java?

Since the static method cannot be overridden in Java, but if you declare the same static method in subclass then that would hide the method from the superclass. It means, if you call that method from subclass then the one in the subclass will be invoked but if you call the same method from superclass then the one in superclass will be invoked. This is known as method hiding in Java

165. Can a class implement more than one interface in Java?

Yes, A class can implement more than one interface in Java e.g. A class can be both Comparable and Serializable at the same time. This is why the interface should be the best use for defining Type as described in Effective Java. This feature allows one class to play a polymorphic role in the program.

166. Can a class extend more than one class in Java?

No, a class can only extend just one more class in Java. Though every class also, by default extend the java.lang.Object class in Java.

167. Can an interface extend more than one interface in Java?

Yes, unlike classes, an interface can extend more than one interface in Java. There are several example of this behaviour in JDK itself e.g. java.util.List interface extends both Collection and Iterable interface to tell that it is a Collection as well as it allows iteration via Iterator

168. What will happen if a class extends two interfaces and they both have a method with same name and signature?

In this case, a conflict will arise because the compiler will not able to link a method call due to ambiguity. You will get a compile time error in Java.

169. Can we pass an object of a subclass to a method expecting an object of the super class?

Yes, you can pass that because subclass and superclass are related to each other by Inheritance which provides IS-A property

170. What is the Liskov substitution principle?

The Liskov substitution principle is one of the five object-oriented design principles, collectively known as SOLID principles. This design principle is L of SOLID acronym. The Liskov substitution principle states that in an object oriented program if a function or method is expecting an object of base class then it should work fine with a derived class object as well. If it cannot function properly with derived class object then the derived class is violating the Liskov Substitution principle.

For example, if a method is expecting a List you can also pass ArrayList or LinkedList and it should work just fine because ArrayList and LinkedList both follow Liskov Substitution Principle, but the java.sql.Date which is a subclass of java.util.Date in Java violates Liskov Substitution Principle because you cannot pass an object of java.sql.Date class to a method which is expecting an object of java.util.Date, Why? because all time-related method will throw java.lang.UnsupportedOperationException.

Here is another example of violating The Liskov Substitution Principle, Square is a special type of Rectangle whose adjacent sides are equal but making Square extending Rectangle violates LSP principle.

171. How to call a method of a subclass, if you are holding an object of the subclass in a reference variable of type superclass?

You can call a method of the subclass by first casting the object hold by reference variable of superclass into the subclass. Once you hold the object in subclass reference type, you can call methods from the subclass.

ABSTRACT

172. Abstract class must have only abstract methods. True or false?

False. Abstract methods can also have concrete methods.

173. Is it compulsory for a class which is declared as abstract to have at least one abstract method?

Not necessarily. Abstract class may or may not have abstract methods.

174. Can we use "abstract" keyword with constructor, Instance Initialization Block and Static Initialization Block?

No. Constructor, Static Initialization Block, Instance Initialization Block and variables cannot be abstract.

175. Can we instantiate a class which does not have even a single abstract method but declared as abstract?

No, we can't instantiate a class once it is declared as abstract even though it does not have abstract methods.

176) Can we declare abstract methods as private? Justify your answer?

No. Abstract methods cannot be private. If abstract methods are allowed to be private, then they will not be inherited to sub class and will not get enhanced.

177) We can't instantiate an abstract class. Then why constructors are allowed in abstract class?

It is because, we can't create objects to abstract classes but we can create objects to their sub classes. From sub class constructor, there will be an implicit call to super class constructor. That's why abstract classes should have constructors. Even if you don't write constructor for your abstract class, compiler will keep default constructor.

178) Can we declare abstract methods as static?
No, abstract methods cannot be static.

179) Can a class contain an abstract class as a member?
Yes, a class can have abstract class as its member.

180) Abstract classes can be nested. True or false?
True. Abstract classes can be nested i.e. an abstract class can have another abstract class as its member.

181) Can we declare abstract methods as synchronized?
No, abstract methods cannot be declared as synchronized. But methods which override abstract methods can be declared as synchronized.

182) Can we declare local inner class as abstract?
Yes. Local inner class can be abstract.

183) Can abstract method declaration include throws clause?
Yes. Abstract methods can be declared with throws clause.

INTERFACE

184) Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

185) What is Interface in Java?

An interface in java is a blueprint of a class. It has static constants and abstract methods only.

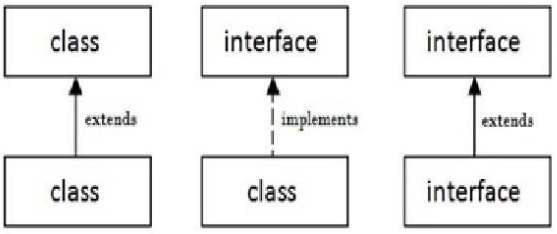
The interface in java is a mechanism to achieve fully abstraction. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.

Java Interface also represents IS-A relationship.

It cannot be instantiated just like abstract class.

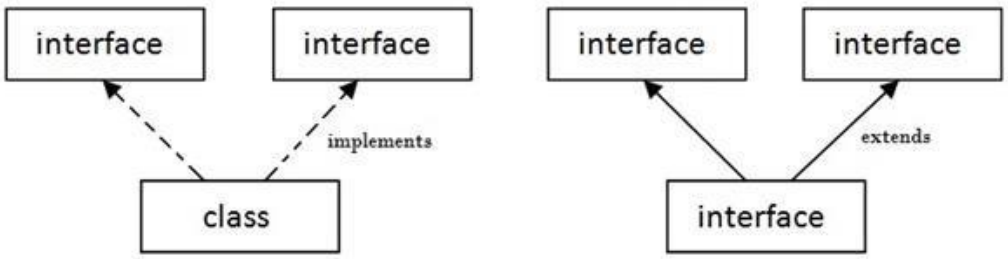
186) What is the relationship between classes and interfaces?

As shown in the figure given below, a class extends another class, an interface extends another interface but a class implements an interface.



187) What is the multiple inheritance in Java by interface?

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance



Example:

```
interface Printable{
void print();
}

interface Showable{
void show();
}

class A implements Printable,Showable{

public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
A obj = new A();
obj.print();
obj.show();
}
}
```

Output: Hello
Welcome

188) Multiple inheritance is not supported through class in java but it is possible by interface, why?

multiple inheritance is not supported in case of class. But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class.

For example:

```
interface Printable{
void print();
}
interface Showable{
void print();
}

class TestTnterface1 implements Printable,Showable
{
public void print()
{
System.out.println("Hello");
}
public static void main(String args[])
{
TestTnterface1 obj = new TestTnterface1();
obj.print();
}
}
```

Hello

As you can see in the above example, Printable and Showable interface have same methods but its implementation is provided by class TestTnterface1, so there is no ambiguity.

189) What is marker or tagged interface?

An interface that have no member is known as marker or tagged interface. For example: Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

```
public interface Serializable
{
}
```

190) What is Nested Interface in Java?

An interface can have another interface known as nested interface.

Example:

```
interface printable{
void print();
interface MessagePrintable{
void msg();
}
}
```

191) What is the meaning of S.O.L.I.D?

As stated above, S.O.L.I.D represents five principles of Java which are:

S: Single responsibility principle

O: Open-closed principle

L: Liskov substitution principle

I: Interface segregation principle

D: Dependency inversion principle

Single Responsibility Principle (SRP)

According to the single responsibility principle, there should be only one reason due to which a class has to be changed. It means that a class should have one task to do. This principle is often termed as subjective. The principle can be well understood with an example. Imagine there is a class which performs following operations:

connect to a database

read some data from database tables
finally, write it to a file.
Have you imagined the scenario? Here the class has multiple reasons to change, and few of them are the modification of file output, new data base adoption. When we are talking about single principle responsibility, we would say, there are too many reasons for the class to change; hence, it doesn't fit properly in the single responsibility principle.

Open Closed Principle

According to open closed principle, entities or objects should remain open for extension, but they should stay closed for modification. To be precise, according to this principle, a class should be written in such a manner that it performs its job flawlessly without the assumption that people in the future will simply come and change it. Hence, the class should remain closed for modification, but it should have the option to get extended. Ways of extending the class include:
1. Inheriting from the class
2. Overwriting the required behaviours from the class
3. Extending certain behaviours of the class
An excellent example of open closed principle can be understood with the help of browsers. Do you remember installing extensions in your chrome browser?
Basic function of chrome browser is to surf different sites. Do you want to check grammar when you are writing an email using chrome browser? If yes, you can simply use Grammar extension, it provides you grammar check on the content.
This mechanism where you are adding things for increasing the functionality of the browser is an extension. Hence, the browser is a perfect example of functionality that is open for extension but is closed for modification. In simple words, you can enhance the functionality by adding/installing plug-in on your browser, but cannot build anything new.

Liskov Substitution Principle

Liskov substitution principle assumes q(x) to be a property, provable about entities of x which belongs to type T. Now, according to this principle, the q (y) should be now provable for objects y that belongs to type S, and the S is actually a subtype of T. Are you now confused and don't know what Liskov substitution principle actually mean? The definition of it might be a bit complex, but in fact, it is quite easy. The only thing is that every subclass or derived class should be substitutable for their parent or base class.
You can say that it is a unique object-oriented principle. The principle can further be simplified by understanding this principle; a child type of a particular parent type without making any complication or blowing things up should have the ability to stand in for that parent. This principle is closely related to Liskov Substitution principle.

Interface Segregation Principle

According to interface segregation principle, a client, no matter what should never be forced to implement an interface that it does not use or the client should never be obliged to depend on any method, which is not used by them.
So basically, the interface segregation principles as you prefer the interfaces, which are small but client specific instead of monolithic and bigger interface.
In short, it would be bad for you to force the client to depend on a certain thing, which they don't need.
Let us now again take an example to understand this.
Let's take a simple example. You are implementing your own ArrayList and LinkedList in java. You create an interface called List which both classes will implement.

```
public interface List<T> {
        public T get();
```

```
	public void add(T t);
	public T poll();
	public T peek();
}
```

Let's create LinkedList class now.

```
public class LinkedList implements List<Integer>{

	@Override
	public Integer get() {
		// Implement this method
		return null;
	}

	@Override
	public void add(Integer t) {
		// Implement this method
	}

	@Override
	public Integer poll() {
		// Implement this method
		return null;
	}

	@Override
	public Integer peek() {
		// Implement this method
		return null;
	}
}

public class ArrayList implements List<Integer>{
```

```
	@Override
	public Integer get() {
		// Implement this method
		return null;
	}

	@Override
	public void add(Integer t) {
		// Implement this method
	}

	@Override
	public Integer poll() {
		// ArrayList does not require this method
		return null;
	}

	@Override
	public Integer peek() {
		// ArrayList does not require this method
		return null;
	}
}
```

Do you see the problem, even though you do not require poll and peek method in ArrayList, we have implemented them.
The correct solution for above problem will be:

```
public interface Deque<T> {
	 public T poll();
	public T peek();
}
```

And remove peek and poll from list interface.

```
public interface List<T> {
	public T get();
	public void add(T t);
}
```

Let's change LinkedList class now.

```
public class LinkedList implements List<Integer>,Deque<Integer>{

	@Override
	public Integer get() {
		// Implement this method
		return null;
	}

	@Override
	public void add(Integer t) {
		// Implement this method
	}

	@Override
	public Integer poll() {
		// Implement this method
		return null;
	}

	@Override
	public Integer peek() {
		// Implement this method
		return null;
	}
}
```

Let's change ArrayList class now.

```
public class ArrayList implements List<Integer>{

	@Override
	public Integer get() {
		// Implement this method
		return null;
	}

	@Override
	public void add(Integer t) {
		// Implement this method
	}
}
```

As you can see, we have segregated two interface to achieve required functionality.

Dependency Inversion Principle

According to this, dependency inversion principle, entities should depend only on abstractions but not on concretions. According to it, the high-level module must never rely on any low-level module but should depend on abstractions. Let us again understand it through another practical example.

You go to a local store to buy something, and you decide to pay for it by using your debit card. So, when you give your card to the clerk for making the payment, the clerk doesn't bother to check what kind of card you have given. Even if you have given a Visa card, he will not put out a Visa machine for swiping your card. The type of credit card or debit card that you have for paying does not even matter; they will simply swipe it. So, in this example, you can see that both you and the clerk are dependent on the credit card abstraction and you are not worried about the specifics of the card. This is what a dependency inversion principle is.

192) Rinku has written the code like below. But, it is showing compile time error. Can you identify what mistake she has done?

```
class X
{
   //Class X Members
}

class Y
{
   //Class Y Members
}

class Z extends X, Y
{
   //Class Z Members
}
```

Answer :

In Java, a class cannot extend more than one class. Class Z is extending two classes – Class X and Class Y. It is a compile time error in java.

193) What will be the output of this program?

```
class A
{
   int i = 10;
}

class B extends A
{
   int i = 20;
}

public class MainClass
{
```

```
    public static void main(String[] args)
    {
        A a = new B();

        System.out.println(a.i);
    }
}
```

Answer :

10

194) What will be the output of the below program?

```
class A
{
    {
        System.out.println(1);
    }
}

class B extends A
{
    {
        System.out.println(2);
    }
}

class C extends B
{
    {
        System.out.println(3);
    }
}

public class MainClass
{
```

```
    public static void main(String[] args)
    {
        C c = new C();
    }
}
```

Answer :

1

2

3

195) Can a class extend itself?

No. A class cannot extend itself.

196) What will be the output of the following program?

```
class A
{
    String s = "Class A";
}

class B extends A
{
    String s = "Class B";

    {
        System.out.println(super.s);
    }
}

class C extends B
{
    String s = "Class C";

    {
        System.out.println(super.s);
```

```
    }
}

public class MainClass
{
    public static void main(String[] args)
    {
        C c = new C();

        System.out.println(c.s);
    }
}
```

Answer :

Class A

Class B

Class C

197) What will be the output of this program?

```
class A
{
    static
    {
        System.out.println("THIRD");
    }
}

class B extends A
{
    static
    {
        System.out.println("SECOND");
    }
}
```

```
class C extends B
{
   static
   {
      System.out.println("FIRST");
   }
}

public class MainClass
{
   public static void main(String[] args)
   {
      C c = new C();
   }
}
```

Answer :

THIRD

SECOND

FIRST

198) What will be the output of the below program?

```
class A
{
   public A()
   {
      System.out.println("Class A Constructor");
   }
}

class B extends A
{
   public B()
   {
```

```
        System.out.println("Class B Constructor");
    }
}

class C extends B
{
    public C()
    {
        System.out.println("Class C Constructor");
    }
}

public class MainClass
{
    public static void main(String[] args)
    {
        C c = new C();
    }
}
```

Answer :

Class A Constructor

Class B Constructor

Class C Constructor

199) Private members of a class are inherited to sub class. True or false?

False. Private members are not inherited to sub class.

200) What will be the output of the following program?

```
class X
{
    static void staticMethod()
    {
        System.out.println("Class X");
```

```
    }
}

class Y extends X
{
    static void staticMethod()
    {
        System.out.println("Class Y");
    }
}

public class MainClass
{
    public static void main(String[] args)
    {
        Y.staticMethod();
    }
}
```

Answer :

Class Y

201) Below code is showing compile time error. Can you suggest the corrections?

```
class X
{
    public X(int i)
    {
        System.out.println(1);
    }
}

class Y extends X
{
```

```
    public Y()
    {
        System.out.println(2);
    }
}
```

Answer :
Write explicit calling statement to super class constructor in Class Y constructor.
Correction code is given below.

```
class X
{
    public X(int i)
    {
        System.out.println(1);
    }
}

class Y extends X
{
    public Y()
    {
        super(10);          //Correction

        System.out.println(2);
    }
}
```

202) What is Polymorphism?

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class's reference is used to refer to a child class object.

203) What is the difference between 'Overloading' and 'Overriding'?

Method overloading increases the readability of the program. On the other hand, method overriding provides the specific implementation of the method that is already provided by its super class. Parameter must be different in case of overloading whereas it must be same in case of overriding.

204) How is Polymorphism supported in Java?

Java has excellent support of polymorphism in terms of 'inheritance', 'method overloading' and 'method overriding'.
'Method overriding' allows Java to invoke method based on a particular object at run-time instead of declared type while coding
Example:

```
public class TradingSystem
{
public String getDescription()
{
return "electronic trading system";
}
}

public class DirectMarketAccessSystem extends TradingSystem{
public String getDescription(){
return "direct market access system";
}
}

public class CommodityTradingSystem extends TradingSystem{
public String getDescription(){
return "Futures trading system";
}
}
```

Here we have a super class called 'TradingSystem' and its two implementations 'DirectMarketAccessSystem' and 'CommodityTradingSystem'. Here we will write a code which is flexible enough to work with. Any future implementation of 'TradingSystem' can be achieved by using Polymorphism in Java.

205) Where can one use Polymorphism in code?

Its common practice to always replace concrete implementation with interface, but it's not that easy and it comes with practice but here are some common places where we can check for polymorphism:

1) Method argument:

Always use super type in method argument that will give you leverage to pass any implementation while invoking method. For example:

```
public void showDescription(TradingSystem tradingSystem)
{
tradingSystem.description();
}
```

2) Variable names:

Always use Super type while you are storing reference returned from any Factory method in Java. This gives you flexibility to accommodate any new implementation from Factory. Here is an example of polymorphism while writing Java code which you can use while retrieving reference from Factory:

```
String systemName = Configuration.getSystemName();
TradingSystem system = TradingSystemFactory.getSystem(systemName);
```

3) Return type of method

Return type of any method is another place where you should be using interface to take advantage of Polymorphism in Java. In fact this is a requirement of Factory design pattern in Java to use interface as return type for factory method.

```
public TradingSystem getSystem(String name){
//code to return appropriate implementation
}
```

206) Explain 'Parametric Polymorphism' in Java.

Java started to support parametric polymorphism with introduction of Generic in JDK1.5. Collection classes in JDK 1.5 are written using Generic Type which allows Collections to hold any type of object in run time without any change in code and this has been achieved by passing actual Type as parameter. For example see the below code of a parametric cache written using Generic which shows use of parametric polymorphism in Java

Example:

```
interface cache{
public void put(K key, V value);
public V get(K key);
}
```

207) Can you override a private or static method in Java?

You cannot override a private or static method in Java. If you create a similar method with same return type and same method arguments in child class then it will hide the super class method; this is known as method hiding. Similarly, you cannot override a private method in sub class because it's not accessible there. What you can do is create another private method with the same name in the child class.

Example:

```
class Base {
private static void display() {
```

```
System.out.println("Static or class method from Base");
}
public void print() {
System.out.println("Non-static or instance method from Base");
}
class Derived extends Base {
private static void display() {
System.out.println("Static or class method from Derived");
}
public void print() {
System.out.println("Non-static or instance method from Derived");
}
public class test {
public static void main(String args[])
{
Base obj= new Derived();
obj1.display();
obj1.print();
}
}
```

208)What is association?

Association is a relationship where all object have their own lifecycle and there is no owner. Let's take an example of Teacher and Student. Multiple students can associate with a single teacher and a single student can associate with multiple teachers but there is no ownership between the objects and both have their own lifecycle. These relationships can be one to one, One to many, many to one and many to many.

209)What do you mean by aggregation?

Aggregation is a specialized form of Association where all object have their own lifecycle but there is ownership and child object cannot belongs to another parent object. Let's take an example of Department and

teacher. A single teacher cannot belong to multiple departments, but if we delete the department teacher object will not destroy.

210) What is composition in Java?

Composition is again specialized form of Aggregation and we can call this as a "death" relationship. It is a strong type of Aggregation. Child object does not have their lifecycle and if parent object deletes all child object will also be deleted. Let's take again an example of relationship between House and rooms. House can contain multiple rooms there is no independent life of room and any room cannot belongs to two different house if we delete the house room will automatically delete.

211) What is the difference between aggregation and composition?

Aggregation represents the weak relationship whereas composition represents the strong relationship. For example, the bike has an indicator (aggregation), but the bike has an engine (composition).

212)What is the use of final keyword in java?

- By using final keyword we can make
- Final class
- Final method
- Final variables
- If we declare any class as final we cannot extend that class
- If we declare any method as final it cannot be overridden in sub class
- If we declare any variable as final its value unchangeable once assigned

213) What is the main difference between abstract method and final method?

Abstract methods must be overridden in sub class where as final methods cannot be overridden in sub class

214) What is the actual use of final class in java?

1. If a class needs some security and it should not participate in inheritance in this scenario we need to use final class.

2. We cannot extend final class.

215) Is it possible to declare final variables without initialization?

1. No. It's not possible to declare a final variable without initial value assigned.
2.
3. While declaring itself we need to initialize some value and that value cannot be change at any time.

216) Can we declare constructor as final?

No. Constructors cannot be final.

217) Can we declare interface as final?

No. We cannot declare interface as final because interface should be implemented by some class so it's not possible to declare interface as final.

218)What will happen if we try to override final methods in sub classes?

Compile time error will come: Cannot override the final method from Super class

219) Can we create object for final class?

Yes we can create object for final class.

220) What is the use of final keyword in java?

final keyword in java is used to make any class or a method or a field as unchangeable. You can't extend a final class, you can't override a final method and you can't change the value of a final field. final keyword is used to achieve high level of security while coding.

221) What is the blank final field?

Uninitialized final field is called blank final field.

222) When do you override hashcode and equals() ?

Whenever necessary especially if you want to do equality check or want to use your object as key in HashMap.

223) Can we change the state of an object to which a final reference variable is pointing?

Yes, we can change the state of an object to which a final reference variable is pointing, but we can't re-assign a new object to this final reference variable.

224) What is the main difference between abstract methods and final methods?

Abstract methods must be overridden in the sub classes and final methods are not at all eligible for overriding.

225) What is the use of final class?

A final class is very useful when you want a high level of security in your application. If you don't want inheritance of a particular class, due to security reasons, then you can declare that class as a final.

226)Can we change the value of an interface field? If not, why?

No, we can't change the value of an interface field. Because interface fields, by default, are final and static. They remain constant for whole execution of a program.

227)Where all we can initialize a final non-static global variable if it is not initialized at the time of declaration?

In all constructors or in any one of instance initialization blocks.

228)What is final class, final method and final variable?

final class is a class that cannot be extended.
final method is a method that cannot be overridden in the sub class.
final variable is a variable that cannot change its value once it is initialized.

229) Where all we can initialize a final static global variable if it is not initialized at the time of declaration?

In any one of static initialization blocks.

230) Can we use non-final local variables inside a local inner class?

No. Only final local variables can be used inside a local inner class.

231)Can we declare constructors as final?

No, constructors cannot be final.

EXCEPTION HANDLING

231. What is an exception?

Exception is an abnormal condition which occurs during the execution of a program and disrupts normal flow of the program. This exception must be handled properly. If it is not handled, program will be terminated abruptly.

232. How the exceptions are handled in java? OR Explain exception handling mechanism in java?

Exceptions in java are handled using try, catch and finally blocks.

try block : The code or set of statements which are to be monitored for exception are kept in this block.

catch block : This block catches the exceptions occurred in the try block.

finally block : This block is always executed whether exception is occurred in the try block or not and occurred exception is caught in the catch block or not.

233. What is the difference between error and exception in java?

Errors are mainly caused by the environment in which an application is running. For example, OutOfMemoryError happens when JVM runs out of memory. Whereas exceptions are mainly caused by the application itself. For example, NullPointerException occurs when an application tries to access null object.

234. Can we keep other statements in between try, catch and finally blocks?

No. We shouldn't write any other statements in between try, catch and finally blocks. They form a one unit.

```
try
{
   // Statements to be monitored for exceptions
}
```

```
//You can't keep statements here

catch(Exception ex)
{
   //Catching the exceptions here
}

//You can't keep statements here

finally
{
   // this block is always executed
}
```

235. Can we write only try block without catch and finally blocks?

No, it shows compilation error. The try block must be followed by either catch or finally block. You can remove either catch block or finally block but not both.

236. There are three statements in a try block – statement1, statement2 and statement3. After that there is a catch block to catch the exceptions occurred in the try block. Assume that exception has occurred in statement2. Does statement3 get executed or not?

No. Once a try block throws an exception, remaining statements will not be executed. Control comes directly to catch block.

237. What is unreachable catch block error?

When you are keeping multiple catch blocks, the order of catch blocks must be from most specific to most general ones i.e. sub classes of Exception must come first and super classes later. If you keep super

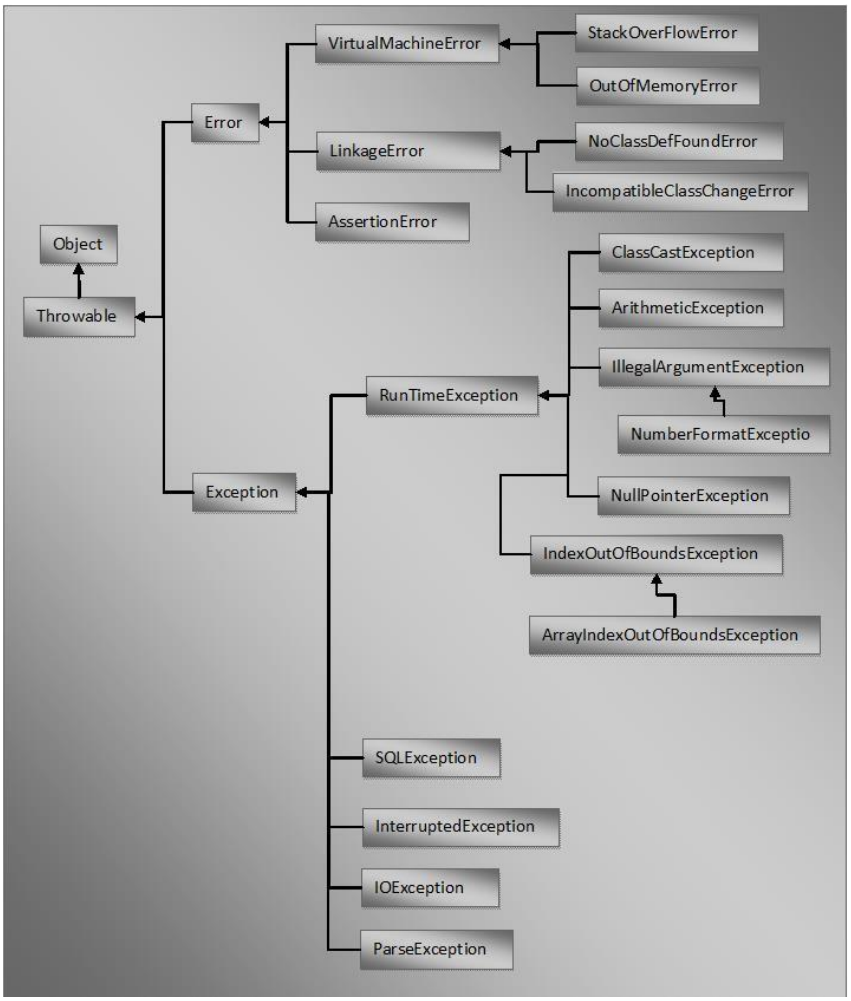
classes first and sub classes later, compiler will show unreachable catch block error.

```
public class ExceptionHandling
{
   public static void main(String[] args)
   {
      try
      {
         int i = Integer.parseInt("abc");   //This statement throws
NumberFormatException
      }

      catch(Exception ex)
      {
         System.out.println("This block handles all exception types");
      }

      catch(NumberFormatException ex)
      {
         //Compile time error
         //This block becomes unreachable as
         //exception is already caught by above catch block
      }
   }
}
```

238. Explain the hierarchy of exceptions in java?



Throwable class is the super class of all the exception types. Below Throwable class there are two subclasses which denotes two distinct branches of exceptions -

Exception - An Exception indicates that a problem has occurred, but it is not a serious system problem. The user programs you write will throw and catch Exceptions.

Error - It defines exceptions that are not expected to be caught by your program. Exceptions of type Error are used by the Java run-time system to indicate errors having to do with the run-time environment, itself. Examples of error are StackOverflowError, OutOfMemoryError etc.

Below Exception there is a distinct subclass RunTimeExcpetion - RunTimeExcpetion and its descendants denote the exceptional conditions

that are external to the application, and the application usually cannot anticipate or recover from them.

239. What are run time exceptions in java. Give example?

The exceptions which occur at run time are called as run time exceptions. These exceptions are unknown to compiler. All sub classes of java.lang.RunTimeException and java.lang.Error are run time exceptions. These exceptions are unchecked type of exceptions. For example, NumberFormatException, NullPointerException, ClassCastException, ArrayIndexOutOfBoundException, StackOverflowError etc.

240. What is OutOfMemoryError in java?

OutOfMemoryError is the sub class of java.lang.Error which occurs when JVM runs out of memory.

241. What are checked and unchecked exceptions in java?

Checked exceptions are the exceptions which are known to compiler. These exceptions are checked at compile time only. Hence the name checked exceptions. These exceptions are also called compile time exceptions. Because, these exceptions will be known during compile time.

Unchecked exceptions are those exceptions which are not at all known to compiler. These exceptions occur only at run time. These exceptions are also called as run time exceptions. All sub classes of java.lang.RunTimeException and java.lang.Error are unchecked exceptions.

242. Can we keep the statements after finally block If the control is returning from the finally block itself?

No, it gives unreachable code error. Because, control is returning from the finally block itself. Compiler will not see the statements after it. That's why it shows unreachable code error.

243. Does finally block get executed If either try or catch blocks are returning the control?

Yes, finally block will be always executed no matter whether try or catch blocks are returning the control or not.

244. Can we throw an exception manually? If yes, how?

Yes, we can throw an exception manually using throw keyword.

Example:

```
try
{
   NumberFormatException ex = new NumberFormatException();
//Creating an object to NumberFormatException explicitly
     throw ex;       //throwing NumberFormatException object explicitly
using throw keyword
}
catch(NumberFormatException ex)
{
   System.out.println("explicitly thrown NumberFormatException object
will be caught here");
}
```

245. What is Re-throwing an exception in java?

Exceptions raised in the try block are handled in the catch block. If it is unable to handle that exception, it can re-throw that exception using throw keyword. It is called re-throwing an exception.

```
try
{
   String s = null;
   System.out.println(s.length()); //This statement throws
NullPointerException
}
catch(NullPointerException ex)
{
   System.out.println("NullPointerException is caught here");
```

```
    throw ex;     //Re-throwing NullPointerException
}
```

246. Why it is always recommended that clean up operations like closing the DB resources to keep inside a finally block?

Because finally block is always executed whether exceptions are raised in the try block or not and raised exceptions are caught in the catch block or not. By keeping the cleanup operations in finally block, you will ensure that those operations will be always executed irrespective of whether exception is occurred or not.

247. Can we override a super class method which is throwing an unchecked exception with checked exception in the sub class?

No. If a super class method is throwing an unchecked exception, then it can be overridden in the sub class with same exception or any other unchecked exceptions but cannot be overridden with checked exceptions.

248. What is StackOverflowError in java?

StackOverflowError is an error which is thrown by the JVM when stack overflows.

249. Can we override a super class method which is throwing an unchecked exception with checked exception in the sub class?

No. If a super class method is throwing an unchecked exception, then it can be overridden in the sub class with same exception or any other unchecked exceptions but cannot be overridden with checked exceptions.

250. Which class is the super class for all types of errors and exceptions in java?

java.lang.Throwable is the super class for all types of errors and exceptions in java.

251. What are the legal combinations of try, catch and finally blocks?

1)

```
try
{
   //try block
}
catch(Exception ex)
{
   //catch block
}
```

2)

```
try
{
   //try block
}
finally
{
   //finally block
}
```

3)

```
try
{
   //try block
}
catch(Exception ex)
{
   //catch block
}
finally
{
```

```
    //finally block
}
```

252. What is the use of printStackTrace() method?

printStackTrace() method is used to print the detailed information about the exception occurred.

253. Give some examples to checked exceptions?

ClassNotFoundException, SQLException, IOException

254. Give some examples to unchecked exceptions?

NullPointerException, ArrayIndexOutOfBoundsException, NumberFormatException

255. What is the difference between Exception and Error in java ?

Exception and Error classes are both subclasses of the Throwable class. The Exception class is used for exceptional conditions that a user's program should catch. The Error class defines exceptions that are not expected to be caught by the user program.

256. What is the difference between throw and throws?

The throw keyword is used to explicitly raise a exception within the program. On the contrary, the throws clause is used to indicate those exceptions that are not handled by a method. Each method must explicitly specify which exceptions does not handle, so the callers of that method can guard against possible exceptions. Finally, multiple exceptions are separated by a comma.

257. How does finally block differ from finalize() method ?

A finally block will be executed whether or not an exception is thrown and is used to release those resources held by the application. Finalize is a protected method of the Object class, which is called by the Java Virtual Machine (JVM) just before an object is garbage collected.

258. What Is A Nested Try Statement?

A try-catch-finally block can reside inside another try-catch-finally block that is known as nested try statement.

```
public class NestedTryDemo {
   public static void main(String[] args) {
      try{
         System.out.println("In Outer try block");
         try{
            System.out.println("In Inner try block");
            int a = 7 / 0;
         }catch (IllegalArgumentException e) {
            System.out.println("IllegalArgumentException caught");
         }finally{
            System.out.println("In Inner finally");
         }
      }catch (ArithmeticException e) {
         System.out.println("ArithmeticException caught");
      }finally {
         System.out.println("In Outer finally");
      }
   }
}
```

259. What Are Multiple Catch Blocks?

There might be a case when a code enclosed with in a try block throws more than one exception. To handle these types of situations, two or more catch clauses can be specified where each catch clause catches a different type of exception. When an exception is thrown, each of the catch statement is inspected in order, and the first one whose type matches that of the thrown exception is executed.

Example : code snippet

```
 int a[] = {0};
 try{
    int b = 7/a[i];
```

```
}catch(ArithmeticException aExp){
    aExp.printStackTrace();
}catch(ArrayIndexOutOfBoundsException aiExp){
    aiExp.printStackTrace();
}
```

260. What Is Exception Propagation?

When an exceptional condition occurs within a method, the method (where the exception occurred) creates an Exception Object and throws it. The created exception object contains information about the error, its type and the state of the program when the error occurred.

The method where the exception is thrown may handle that exception itself or pass it on. In case it passes it on, run time system goes through the method hierarchy that had been called to get to the current method to search for a method that can handle the exception.

If your program is not able to catch any particular exception, that will ultimately be processed by the default handler. This process of going through the method stack is known as Exception propagation.

261. Compare Final Vs Finally Vs Finalize

final - final keyword is used to restrict in some way. It can be used with variables, methods and classes. When a variable is declared as final, its value cannot be changed once it is initialized. Except in case of blank final variable, which must be initialized in the constructor.
If you make a method final in Java, that method can't be overridden in a sub class.
If a class is declared as final then it cannot be sub classed.

finally - finally is part of exception handling mechanism in Java. finally block is used with try-catch block. finally block is always executed whether any exception is thrown or not and raised exception is handled in catch block or not. Since finally block always executes thus it is primarily

used to close the opened resources like database connection, file handles etc.

finalize() - finalize() method is a protected method of java.lang.Object class. Since it is in Object class thus it is inherited by every class. This method is called by garbage collector thread before removing an object from the memory. This method can be overridden by a class to provide any cleanup operation and gives object final chance to cleanup before getting garbage collected.

```
protected void finalize() throws Throwable
{
   //resource clean up operations
}
```

262. What Are The Rules Of Exception Handling With Respect To Method Overriding?

There are certain restrictions while overriding a method in case of exception handling in Java.

1.If super class method has not declared any exception using throws clause then subclass overridden method can't declare any checked exception though it can declare unchecked exception.

2.If super class method has declared an exception using throws clause then subclass overridden method can do one of the three things.

a) sub-class can declare the same exception as declared in the super-class method.

b)subclass can declare the subtype exception of the exception declared in the super class method.

c) But subclass method cannot declare any exception that is up in the hierarchy than the exception declared in the super class method. subclass method can choose not to declare any exception at all.

263. What Is The Error In The Following Code?

```
class Parent{
  public void displayMsg() throws IOException{
```

```
    System.out.println("In Parent displayMsg()");
   throw new IOException("Problem in method - displayMsg - Parent");
  }
}
public class ExceptionOverrideDemo extends Parent
{
public void displayMsg() throws Exception
{
System.out.println("In ExceptionOverrideDemo displayMsg()");
throw new Exception("Problem in method - displayMsg -
ExceptionOverrideDemo");
}
}
```

Here parent class had declared IOException where as subclass has declared Exception. Exception is the super class of IOException thus it is wrong according to the rules of method overriding and exception handling. Thus the code will give compiler error.

264. What Is Multi-catch Statement in Java 7?

Before Java 7 multi-catch statement, if two or more exceptions were handled in the same way, we still had to write separate catch blocks for handling them.

```
 catch(IOException exp){
  logger.error(exp);
  throw exp;
 }catch(SQLException exp){
  logger.error(exp);
  throw exp;
 }
```

With Java 7 and later it is possible to catch multiple exceptions in one catch block, which eliminates the duplicated code. Each exception type within the multi-catch statement is separated by Pipe symbol (|).

```
catch(IOException | SQLException exp){
   logger.error(exp);
   throw exp;
}
```

265. What Is Try-with-resources Or Arm in Java 7?

Java 7 introduced a new form of try known as try-with-resources for Automatic Resource Management (ARM). Here resource is an object that must be closed after the program is finished with it. Example of resources would be an opened file handle or database connection etc.

Before the introduction of try-with-resources we had to explicitly close the resources once the try block completes normally or abruptly.

```
try {
   br = new BufferedReader(new FileReader("C:\test.txt"));
   System.out.println(br.readLine());
} catch (IOException e) {
   e.printStackTrace();
} finally {
   try {
      if (br != null){
         System.out.println("Closing the file");
         br.close();
      }

   } catch (IOException ex) {
      ex.printStackTrace();
   }
}
```

try-with-resources helps in reducing such boiler plate code. Let's see the same example using try-with-resources.

```
try(BufferedReader br = new BufferedReader(new
FileReader("C:\test.txt"))) {
   System.out.println(br.readLine());
} catch (IOException e) {
   e.printStackTrace();
}
```

266. When Is Custom Exception Class Needed? How to Create a Custom Exception Class?

According to Java Docs, you should write your own exception classes if you answer yes to any of the following questions; otherwise, you can probably use someone else's.

1. Do you need an exception type that isn't represented by those in the Java platform?
2. Would it help users if they could differentiate your exceptions from those thrown by classes written by other vendors?
3. Does your code throw more than one related exception?
4. If you use someone else's exceptions, will users have access to those exceptions? A similar question is, should your package be independent and self-contained?

267. What is exception matching?

Exception matching is the process by which the jvm finds out the matching catch block for the exception thrown from the list of catch blocks. When an exception is thrown, Java will try to find by looking at the available catch clauses in the top down manner. If it doesn't find one, it will search for a handler for a supertype of the exception. If it does not find a catch clause that matches a supertype for the exception, then the exception is propagated down the call stack. This process is called as exception matching.

268. What happens if a method does not throw a checked exception directly but calls a method that does? What does 'Ducking' the exception mean?

If a method does not throw a checked exception directly but calls a method that throws an exception then the calling method must handle the 'throw' exception or declare the exception in its 'throws' clause. If the calling method does not handle it and declares the exception, the exception is passed to the next method in the method stack. This is called as ducking the exception down the method stack. e.g. The code below will not compile as the getCar() method has not declared the 'CarNotFoundException' which is thrown by the getColor () method.

```
void getCar()
{
getColor();
}
void getColor()
{
throw new CarNotFoundException();
}
//Fix for the above code is
void getCar() throws CarNotFoundException
{
getColor();
}
void getColor()
{
throw new CarNotFoundException();
}
```

269. Is an empty catch block legal?

An empty catch block is considered legal by leaving the catch block without writing any actual code to handle the exception caught.

MULTITHREADING

270. What is Thread in java?

1. Threads are light weight process.
2. Threads consumes CPU in best possible manner, hence enables multi processing. Multi threading reduces idle time of CPU which improves performance of application.
3. A thread class belongs to java.lang package.
4. We can create multiple threads in java, even if we don't create any Thread, one Thread at least do exist i.e. main thread.
5. Multiple threads run parallel in java.
6. Threads have their own stack.

271. What is difference between Process and Thread in java?

One process can have multiple Threads,
Thread is subdivision of Process. One or more Threads run in the context of process. Threads can execute any part of process. And same part of process can be executed by multiple Threads.
Processes have their own copy of the data segment of the parent process while Threads have direct access to the data segment of its process.

Processes have their own address while Threads share the address space of the process that created it.
Process creation needs whole lot of stuff to be done, we might need to copy whole parent process, but Thread can be easily created.
Processes can easily communicate with child processes but inter-process communication is difficult. While, Threads can easily communicate with other threads of the same process using wait() and notify() methods.
In process all threads share system resource like heap Memory etc. while Thread has its own stack.

Any change made to process does not affect child processes, but any change made to thread can affect the behaviour of the other threads of the process.

272. How to implement Threads in java?

Threads can be created in two ways i.e. by

1. implementing java.lang.Runnable interface or
2. extending java.lang.Thread class and then overriding run method.

Thread has its own variables and methods, it lives and dies on the heap. But a thread of execution is an individual process that has its own call stack. Thread is a lightweight process in java.

Thread creation by implementingjava.lang.Runnableinterface.

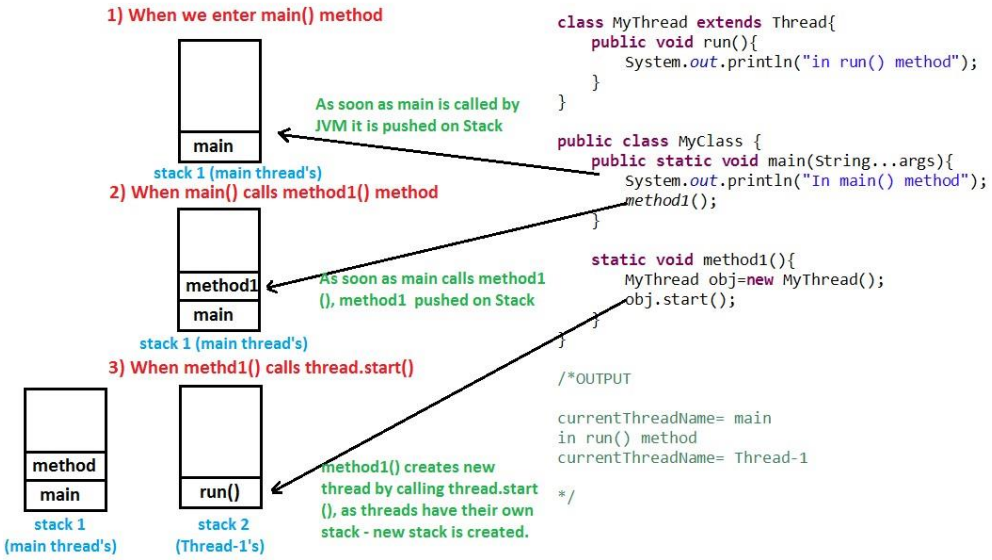
We will create object of class which implements Runnable interface:
Example:
MyRunnable runnable=new MyRunnable();
Thread thread=new Thread(runnable);

create Thread object by calling constructor and passing reference of Runnable interface i.e. runnable object :

example:
Thread thread=new Thread(runnable);

273. Does Thread implements their own Stack, if yes how?

Yes, Threads have their own stack. This is very interesting question, where interviewer tends to check your basic knowledge about how threads internally maintain their own stacks.



274. We should implement Runnable interface or extend Thread class. What are differences between implementing Runnable and extending Thread?

Well the answer is you must extend Thread only when you are looking to modify run() and other methods as well.
If you are simply looking to modify only the run() method implementing Runnable is the best option (Runnable interface has only one abstract method i.e. run()).

Differences between implementing Runnable interface and extending Thread class –

Multiple inheritance in not allowed in java: When we implement Runnable interface we can extend another class as well, but if we extend Thread class we cannot extend any other class because java does not allow multiple inheritance.
So, same work is done by implementing Runnable and extending Thread but in case of implementing Runnable we are still left with option of extending some other class. So, it's better to implement Runnable.

Thread safety: When we implement Runnable interface, same object is shared amongst multiple threads, but when we extend Thread class each and every thread gets associated with new object.
Inheritance (Implementing Runnable is lightweight operation) : When we extend Thread unnecessary all Thread class features are inherited, but when we implement Runnable interface no extra feature are inherited, as Runnable only consists only of one abstract method i.e. run() method. So, implementing Runnable is lightweight operation.
Coding to interface: Even java recommends coding to interface. So, we must implement Runnable rather than extending thread. Also, Thread class implements Runnable interface.
Don't extend unless you want to modify fundamental behaviour of class, Runnable interface has only one abstract method i.e. run() : We must extend Thread only when you are looking to modify run() and other methods as well. If you are simply looking to modify only the run() method implementing Runnable is the best option (Runnable interface has only one abstract method i.e. run()). We must not extend Thread class unless we're looking to modify fundamental behaviour of Thread class.
Flexibility in code when we implement Runnable: When we extend Thread first a fall all thread features are inherited and our class becomes direct subclass of Thread, so whatever action we are doing is in Thread class. But, when we implement Runnable we create a new thread and pass runnable object as parameter, we could pass runnable object to ExecutorService & much more. So, we have more options when we implement Runnable and our code becomes more flexible.
ExecutorService : If we implement Runnable, we can start multiple thread created on runnable object with ExecutorService (because we can start Runnable object with new threads), but not in the case when we extend Thread (because thread can be started only once).

275. How can you say Thread behaviour is unpredictable?

The solution to question is quite simple, Thread behaviour is unpredictable because execution of Threads depends on Thread scheduler, thread scheduler may have different implementation on different platforms like windows, Unix etc. Same threading program may produce different output in subsequent executions even on same platform.

To achieve we are going to create 2 threads on same Runnable Object, create for loop in run() method and start both threads. There is no surety that which threads will complete first, both threads will enter anonymously in for loop.

276. When threads are not lightweight process in java?

Threads are lightweight process only if threads of same process are executing concurrently. But if threads of different processes are executing concurrently then threads are heavy weight process.

277. How can you ensure all threads that started from main must end in order in which they started and also main should end in last?

Interviewers tend to know interviewees knowledge about Thread methods. So this is time to prove your point by answering correctly. We can use join() method to ensure all threads that started from main must end in order in which they started and also main should end in last. In other words waits for this thread to die. Calling join() method internally calls join(0);

278. What is difference between starting thread with run() and start() method?

This is quite interesting question, it might confuse you a bit and at time may make you think is there really any difference between starting thread with run() and start() method.

When you call start() method, main thread internally calls run() method to start newly created Thread, so run() method is ultimately called by newly created thread.

When you call run() method main thread rather than starting run() method with newly thread it start run() method by itself.

279. What is significance of using volatile keyword?

Java allows threads to access shared variables. As a rule, to ensure that shared variables are consistently updated, a thread should ensure that it has exclusive use of such variables by obtaining a lock that enforces mutual exclusion for those shared variables.

If a field is declared volatile, in that case the Java memory model ensures that all threads see a consistent value for the variable.

280. What are the Differences between synchronized and volatile keyword in Java?

Volatile can be used as a keyword against the variable, we cannot use volatile against method declaration.
volatile void method1(){} //it's illegal, compilation error.

While synchronization can be used in method declaration or we can create synchronization blocks (In both cases thread acquires lock on object's monitor). Variables cannot be synchronized.
Synchronized method:

synchronized void method2(){} //legal

Synchronized block:

```
void method2(){

synchronized (this) {

//code inside synchronized block.

}
```

}

Synchronized variable (illegal):

synchronized int i;//it's illegal, compilation error.

1. Volatile does not acquire any lock on variable or object, but Synchronization acquires lock on method or block in which it is used
2. Volatile variables are not cached, but variables used inside synchronized method or block are cached.
3. When volatile is used will never create deadlock in program, as volatile never obtains any kind of lock. But in case if synchronization is not done properly, we might end up creating deadlock in program.
4. Synchronization may cost us performance issues, as one thread might be waiting for another thread to release lock on object. But volatile is never expensive in terms of performance.

281. Can you again start Thread?

No, we cannot start a thread again, doing so will throw RuntimeException java.lang.IllegalThreadStateException. The reason is once run() method is executed by Thread, it goes into dead state.
Let's take an example-
Thinking of starting thread again and calling start() method on it (which internally is going to call run() method) for us is somewhat like asking dead man to wake up and run. As, after completing his life person goes to dead state.

282. What is race condition in multithreading and how can we solve it?

When more than one thread try to access same resource without synchronization causes race condition.

So we can solve race condition by using either synchronized block or synchronized method. When no two threads can access same resource at a time phenomenon is also called as mutual exclusion.

283. How threads communicate between each other?

Threads can communicate with each other by using wait(), notify() and notifyAll() methods.

284. Why wait(), notify() and notifyAll() are in Object class and not in Thread class?

1) Every Object has a monitor, acquiring that monitors allow thread to hold lock on object. But Thread class does not have any monitors.

2) wait(), notify() and notifyAll()are called on objects only
When wait() method is called on object by thread it waits for another thread on that object to release object monitor by calling notify() or notifyAll() method on that object.

When notify() method is called on object by thread it notifies all the threads
which are waiting for that object monitor that object monitor is available now.
So, this shows that wait(), notify() and notifyAll() are called on objects only.

Note:
1) wait(), notify() and notifyAll() method being in Object class allows all the threads created on that object to communicate with other. .
2) As multiple threads exists on same object. Only one thread can hold object monitor at a time. As a result thread can notify other threads of same object that lock is available now. But, thread having these methods does not make any sense because multiple threads exists on object its not other way around (i.e. multiple objects exists on thread).

3) Now let's discuss one hypothetical scenario, what will happen if Thread class contains wait(), notify() and notifyAll() methods?
Having wait(), notify() and notifyAll() methods means Thread class also must have their monitor.
Every thread having their monitor will create few problems -
a) Thread communication problem.
b) Synchronization on object won't be possible- Because object has monitor, one object can have multiple threads and thread hold lock on object by holding object monitor. But if each thread will have monitor, we won't have any way of achieving synchronization.
c) Inconsistency in state of object (because synchronization won't be possible).

285. Is it important to acquire object lock before calling wait(), notify() and notifyAll()?

Yes, it's mandatory to acquire object lock before calling these methods on object. As discussed above wait(), notify() and notifyAll() methods are always called from Synchronized block only, and as soon as thread enters synchronized block it acquires object lock (by holding object monitor). If we call these methods without acquiring object lock i.e. from outside synchronize block then java.lang.IllegalMonitorStateException is thrown at runtime.

wait() method needs to enclosed in try-catch block, because it throws compile time exception i.e. InterruptedException.

286. What is deadlock in multithreading? Write a program to form DeadLock in multi threading and also how to solve DeadLock situation. What measures you should take to avoid deadlock?

Deadlock is a situation where two threads are waiting for each other to release lock held by them on resources.

Deadlock Formation Example

```
public class TestThread
 {
  public static Object Lock1 = new Object();
  public static Object Lock2 = new Object();

  public static void main(String args[]) {
    ThreadDemo1 T1 = new ThreadDemo1();
    ThreadDemo2 T2 = new ThreadDemo2();
    T1.start();
    T2.start();
  }

  private static class ThreadDemo1 extends Thread {
    public void run() {
      synchronized (Lock1) {
        System.out.println("Thread 1: Holding lock 1...");

        try { Thread.sleep(10); }
        catch (InterruptedException e) {}
        System.out.println("Thread 1: Waiting for lock 2...");

        synchronized (Lock2)
{
          System.out.println("Thread 1: Holding lock 1 & 2...");
        }
      }
    }
  }
  private static class ThreadDemo2 extends Thread {
    public void run() {
      synchronized (Lock2)  {
        System.out.println("Thread 2: Holding lock 2...");
```

```
          try { Thread.sleep(10); }
          catch (InterruptedException e) {}
          System.out.println("Thread 2: Waiting for lock 1...");

          synchronized (Lock1)
          {
            System.out.println("Thread 2: Holding lock 1 & 2...");
          }
        }
      }
    }
```

When you compile and execute the above program, you find a deadlock situation

Output

```
Thread 1: Holding lock 1...
Thread 2: Holding lock 2...
Thread 1: Waiting for lock 2...
Thread 2: Waiting for lock 1...
```

Deadlock Solution

Example:

```
public class TestThread {
  public static Object Lock1 = new Object();
  public static Object Lock2 = new Object();

  public static void main(String args[]) {
    ThreadDemo1 T1 = new ThreadDemo1();
    ThreadDemo2 T2 = new ThreadDemo2();
    T1.start();
    T2.start();
  }

  private static class ThreadDemo1 extends Thread {
```

```
    public void run() {
      synchronized (Lock1) {
        System.out.println("Thread 1: Holding lock 1...");

        try {
          Thread.sleep(10);
        } catch (InterruptedException e) {}
        System.out.println("Thread 1: Waiting for lock 2...");

        synchronized (Lock2) {
          System.out.println("Thread 1: Holding lock 1 & 2...");
        }
      }
    }
  }
  private static class ThreadDemo2 extends Thread {
    public void run() {
      synchronized (Lock1)  {
        System.out.println("Thread 2: Holding lock 1...");

        try {
          Thread.sleep(10);
        } catch (InterruptedException e) {}
        System.out.println("Thread 2: Waiting for lock 2...");

        synchronized (Lock2) {
          System.out.println("Thread 2: Holding lock 1 & 2...");
        }
      }
    }
  }
}
```

So just changing the order of the locks prevents the program in going into a deadlock situation and completes with the following result

Output:
Thread 1: Holding lock 1...
Thread 1: Waiting for lock 2...
Thread 1: Holding lock 1 & 2...
Thread 2: Holding lock 1...
Thread 2: Waiting for lock 2...
Thread 2: Holding lock 1 & 2...

287. What is life cycle of Thread, explain thread states?

Threads have following states -
New
Runnable
Running
Waiting/blocked/sleeping
Terminated (Dead)

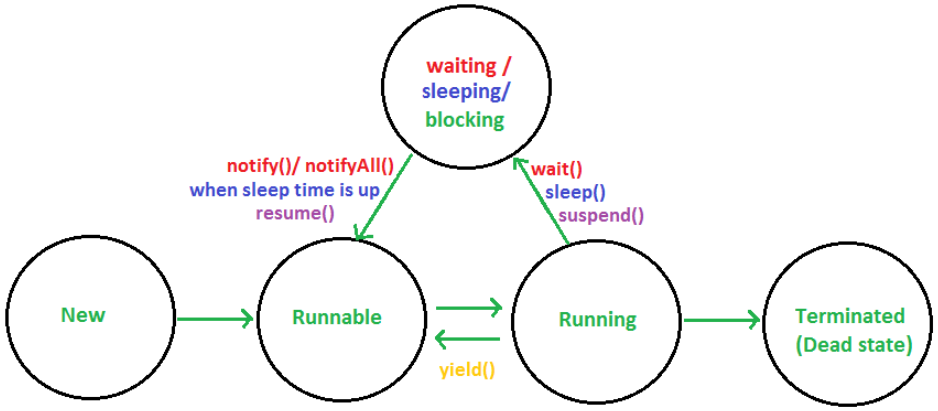


Fig. THREAD STATES

Note:

1. Runnable : When start() method is called on thread it enters runnable state.

2. Running : Thread scheduler selects thread to go from runnable to running state. In running state Thread starts executing by entering run() method.

3. Waiting/blocked/sleeping : In this state a thread is not eligible to run.

4. By calling wait()method thread go from running to waiting state. In waiting state it will wait for other threads to release object monitor/lock.

5. By calling sleep() method, thread goes from running to sleeping state. In sleeping state it will wait for sleep time to get over.

6. Terminated (Dead) : A thread is considered dead when its run() method completes.

288. Are you aware of pre-emptive scheduling and time slicing?

In pre-emptive scheduling, the highest priority thread executes until it enters into the waiting or dead state.
In time slicing, a thread executes for a certain predefined time and then enters runnable pool. Then thread can enter running state when selected by thread scheduler.

289. What are daemon threads?

Daemon threads are low priority threads which run intermittently in background for doing garbage collection.

Features of daemon() threads

1. Thread scheduler schedules these threads only when CPU is idle.
2. Daemon threads are service oriented threads, they serves all other threads.
3. These threads are created before user threads are created and die after all other user threads dies.
4. Priority of daemon threads is always 1 (i.e. MIN_PRIORITY).
5. User created threads are non daemon threads.

6. JVM can exit when only daemon threads exist in system.
7. We can use isDaemon() method to check whether thread is daemon thread or not.
8. We can use setDaemon(boolean on) method to make any user method a daemon thread.
9. If setDaemon(boolean on) is called on thread after calling start() method than IllegalThreadStateException is thrown.
10. You may like to see how daemon threads work, for that you can use VisualVM or jStack.
11. I have provided Thread dumps over there which shows daemon threads which were intermittently running in background.

290. What is significance of yield() method, what state does it put thread in?

yield() is a native method its implementation in java 6 has been changed as compared to its implementation java 5. As method is native its implementation is provided by JVM

In java 5, yield() method internally used to call sleep() method giving all the other threads of same or higher priority to execute before yielded thread by leaving allocated CPU for time gap of 15 millisecond.

But java 6, calling yield() method gives a hint to the thread scheduler that the current thread is willing to yield its current use of a processor. The thread scheduler is free to ignore this hint. So, sometimes even after using yield() method, you may not notice any difference in output.

Note:
1. yield() method when called on thread gives a hint to the thread scheduler that the current thread is willing to yield its current use of a processor. The thread scheduler is free to ignore this hint.
2. Thread state: when yield() method is called on thread it goes from running to runnable state, not in waiting state. Thread is eligible to run but not running and could be picked by scheduler at anytime.

3. Waiting time: yield() method stops thread for unpredictable time.
4. Static method: yield()is a static method, hence calling Thread.yield() causes currently executing thread to yield.
5. Native method: implementation of yield() method is provided by JVM.

291. What is the Difference between wait() and sleep() ?

1. wait() method is always called from synchronized block i.e. wait() method needs to lock object monitor before object on which it is called. But sleep() method can be called from outside synchronized block i.e. sleep() method doesn't need any object monitor.

2. If wait() method is called without acquiring object lock than IllegalMonitorStateException is thrown at runtime, but sleep() method never throws such exception

3. wait() method belongs to java.lang.Object class but sleep() method belongs to java.lang.Thread class.

4. wait() method is called on objects but sleep() method is called on Threads not objects.

5. when wait() method is called on object, thread that held object's monitor goes from running to waiting state and can return to runnable state only when notify() or notifyAll()method is called on that object. And later thread scheduler schedules that thread to go from runnable to running state.

When sleep() is called on thread it goes from running to waiting state and can return to runnable state when sleep time is up.

292. Mention some guidelines to write thread safe code

In multithreading environment it's very important to write thread safe code, thread unsafe code can cause a major threat to your application

1. If method is exposed in multithreading environment and it's not synchronized (thread unsafe) than it might lead us to race condition, we must try to use synchronized block and synchronized methods. Multiple threads may exist on same object but only one thread of that object can enter synchronized method at a time, though threads on different object can enter same method at same time.
2. Even static variables are not thread safe, they are used in static methods and if static methods are not synchronized then thread on same or different object can enter method concurrently. Multiple threads may exist on same or different objects of class but only one thread can enter static synchronized method at a time, we must consider making static methods as synchronized.
3. If possible, try to use volatile variables. If a field is declared volatile all threads see a consistent value for the variable. Volatile variables at times can be used as alternate to synchronized methods as well.
4. Final variables are thread safe because once assigned some reference of object they cannot point to reference of other object.

293. How thread can enter waiting, sleeping and blocked state and how can they go to runnable state?

By calling wait()method thread go from running to waiting state. In waiting state it will wait for other threads to release object monitor/lock.

Once notify() or notifyAll()method is called object monitor/lock becomes available and thread can again return to runnable state.

By calling sleep() method, thread go from running to sleeping state. In sleeping state it will wait for sleep time to get over. Once specified sleep time is up thread can again return to runnable state.

suspend() method can be used to put thread in waiting state and resume() method is the only way which could put thread in runnable state.

Thread also may go from running to waiting state if it is waiting for some I/O operation to take place. Once input is available thread may return to running state.

When threads are in running state, yield() method can make thread to go in Runnable state

294. Does thread leaves object lock when sleep() method is called?

When sleep() method is called Thread does not leaves object lock and goes from running to waiting state. Thread waits for sleep time to over and once sleep time is up it goes from waiting to runnable state.

295. Does thread leaves object lock when wait() method is called?

When wait() method is called Thread leaves the object lock and goes from running to waiting state. Thread waits for other threads on same object to call notify() or notifyAll() and once any of notify() or notifyAll() is called it goes from waiting to runnable state and again acquires object lock.

296. What will happen if we don't override run method?

When we call start() method on thread, it internally calls run() method with newly created thread. So, if we don't override run() method newly created thread won't be called and nothing will happen.

297. What will happen if we override start method?

When we call start() method on thread, it internally calls run() method with newly created thread. So, if we override start() method, run() method will not be called until we write code for calling run() method.

298. Suppose you have thread and it is in synchronized method and now can thread enter other synchronized method from that method?

Yes, here when thread is in synchronized method it must be holding lock on object's monitor and using that lock thread can enter other synchronized method

```
class Runnable1 implements Runnable
{

   @Override
   public void run(){
         m1();
   }

   synchronized void m1()
{
         System.out.println("synchronized method1() started");
         m2();
         System.out.println("synchronized method1() ended");
   }

   synchronized void m2()
   {
         System.out.println("in synchronized method2()");
   }

}
 public class Demo {
   public static void main(String args[]) throws InterruptedException{

         Runnable1 mRunnable1=new Runnable1();
         Thread thread1=new Thread(Runnable1,"Thread-1");
         thread1.start();

   }

}
```

OUTPUT

```
synchronized method1() started
in synchronized method2()
synchronized method1() ended
```

299. Give the solution to producer consumer problem

Following example demonstrates how to solve the producer consumer problem using thread.

```
public class ProducerConsumerTest {
  public static void main(String[] args) {
    CubbyHole c = new CubbyHole();
    Producer p1 = new Producer(c, 1);
    Consumer c1 = new Consumer(c, 1);
    p1.start();
    c1.start();
  }
}
class CubbyHole {
  private int contents;
  private boolean available = false;

  public synchronized int get() {
    while (available == false) {
      try {
        wait();
      } catch (InterruptedException e) {}
    }
    available = false;
    notifyAll();
    return contents;
  }
  public synchronized void put(int value) {
```

```
    while (available == true) {
      try {
        wait();
      } catch (InterruptedException e) { }
    }
    contents = value;
    available = true;
    notifyAll();
  }
}
class Consumer extends Thread {
  private CubbyHole cubbyhole;
  private int number;

  public Consumer(CubbyHole c, int number) {
    cubbyhole = c;
    this.number = number;
  }
  public void run() {
    int value = 0;
    for (int i = 0; i < 10; i++) {
      value = cubbyhole.get();
      System.out.println("Consumer #" + this.number + " got: " + value);
    }
  }
}
class Producer extends Thread {
  private CubbyHole cubbyhole;
  private int number;
  public Producer(CubbyHole c, int number) {
    cubbyhole = c;
    this.number = number;
  }
  public void run() {
```

```
    for (int i = 0; i < 10; i++) {
      cubbyhole.put(i);
      System.out.println("Producer #" + this.number + " put: " + i);
      try {
        sleep((int)(Math.random() * 100));
      } catch (InterruptedException e) { }
    }
  }
}
```

```
Producer #1 put: 0
Consumer #1 got: 0
Producer #1 put: 1
Consumer #1 got: 1
Producer #1 put: 2
Consumer #1 got: 2
Producer #1 put: 3
Consumer #1 got: 3
Producer #1 put: 4
Consumer #1 got: 4
Producer #1 put: 5
Consumer #1 got: 5
Producer #1 put: 6
Consumer #1 got: 6
Producer #1 put: 7
Consumer #1 got: 7
Producer #1 put: 8
Consumer #1 got: 8
Producer #1 put: 9
Consumer #1 got: 9
```

300. How to display the status of the Thread?

Following example demonstrates how to display different status of thread using isAlive() & getStatus() methods of Thread.

```
class MyThread extends Thread {
  boolean waiting = true;
  boolean ready = false;
  MyThread() {
  }
  public void run() {
    String thrdName = Thread.currentThread().getName();
    System.out.println(thrdName + " starting.");

    while(waiting) System.out.println("waiting:"+waiting);
    System.out.println("waiting...");
    startWait();
    try {
      Thread.sleep(1000);
    } catch(Exception exc) {
      System.out.println(thrdName + " interrupted.");
    }
    System.out.println(thrdName + " terminating.");
  }
  synchronized void startWait() {
    try {
      while(!ready) wait();
    } catch(InterruptedException exc) {
      System.out.println("wait() interrupted");
    }
  }
  synchronized void notice() {
    ready = true;
    notify();
  }
}
public class Main {
  public static void main(String args[]) throws Exception {
```

```
    MyThread thrd = new MyThread();
    thrd.setName("MyThread #1");
    showThreadStatus(thrd);

    thrd.start();
    Thread.sleep(50);
    showThreadStatus(thrd);

    thrd.waiting = false;
    Thread.sleep(50);
    showThreadStatus(thrd);

    thrd.notice();
    Thread.sleep(50);
    showThreadStatus(thrd);

    while(thrd.isAlive())
    System.out.println("alive");
    showThreadStatus(thrd);
  }
  static void showThreadStatus(Thread thrd) {
    System.out.println(thrd.getName()+" Alive:"+thrd.isAlive()+" State:" +
thrd.getState() );
  }
}
```

Output:

```
MyThread #1 Alive:false State:NEW
MyThread #1 starting.
waiting:true
waiting:true
alive
alive
MyThread #1 terminating.
```

alive
MyThread #1 Alive:false State:TERMINATED

301. How to interrupt a running Thread?

Following example demonstrates how to interrupt a running thread interrupt() method of thread and check if a thread is interrupted using isInterrupted() method.

```
public class GeneralInterrupt extends Object implements Runnable {
  public void run() {
    try {
      System.out.println("in run() - about to work2()");
      work2();
      System.out.println("in run() - back from work2()");
    } catch (InterruptedException x) {
      System.out.println("in run() - interrupted in work2()");
      return;
    }
    System.out.println("in run() - doing stuff after nap");
    System.out.println("in run() - leaving normally");
  }
  public void work2() throws InterruptedException {
    while (true) {
      if (Thread.currentThread().isInterrupted()) {
        System.out.println("C isInterrupted()="+
Thread.currentThread().isInterrupted());
        Thread.sleep(2000);
        System.out.println("D isInterrupted()="+
Thread.currentThread().isInterrupted());
      }
    }
  }
  public void work() throws InterruptedException {
    while (true) {
```

```
        for (int i = 0; i < 100000; i++) {
          int j = i * 2;
        }
        System.out.println("A isInterrupted()="+
Thread.currentThread().isInterrupted());
        if (Thread.interrupted()) {
          System.out.println("B isInterrupted()="+
Thread.currentThread().isInterrupted());
          throw new InterruptedException();
        }
      }
    }
    public static void main(String[] args) {
      GeneralInterrupt si = new GeneralInterrupt();
      Thread t = new Thread(si);
      t.start();
      try {
        Thread.sleep(2000);
      } catch (InterruptedException x) { }

      System.out.println("in main() - interrupting other thread");
      t.interrupt();
      System.out.println("in main() - leaving");
    }
}
```

Output:

```
in run() - about to work2()
in main() - interrupting other thread
in main() - leaving
C isInterrupted()=true
in run() - interrupted in work2()
```

302. What is ThreadPool?

ThreadPool is a pool of threads which reuses a fixed number of threads to execute tasks.

At any point, at most n threads will be active processing tasks. If additional tasks are submitted when all threads are active, they will wait in the queue until a thread is available.

Note:
Advantage of ThreadPool:

Instead of creating new thread every time for executing tasks, we can create ThreadPool which reuses a fixed number of threads for executing tasks.

As threads are reused, performance of our application improves drastically.

303. Can a constructor be synchronized?

No, constructor cannot be synchronized. Because constructor is used for instantiating object, when we are in constructor object is under creation. So, until object is not instantiated it does not need any synchronization. Enclosing constructor in synchronized block will generate compilation error.
And the error is-"Illegal modifier for the constructor, only public, protected & private are permitted"

304. What are thread priorities?

Thread Priority range is from 1 to 10.
Where 1 is minimum priority and 10 is maximum priority.
Thread class provides variables of final static int type for setting thread priority.
publicfinalstaticintMIN_PRIORITY= 1;

publicfinalstaticintNORM_PRIORITY= 5;
publicfinalstaticintMAX_PRIORITY= 10;

Thread with MAX_PRIORITY is likely to get more CPU as compared to low priority threads. But occasionally low priority thread might get more CPU. Because thread scheduler schedules thread on discretion of implementation and thread behaviour is totally unpredictable.
Thread with MIN_PRIORITY is likely to get less CPU as compared to high priority threads. But occasionally high priority thread might less CPU. Because thread scheduler schedules thread on discretion of implementation and thread behaviour is totally unpredictable.

setPriority()method is used for Changing the priority of thread.
getPriority()method returns the thread's priority.

COLLECTIONS AND GENERICS

305. What is the Collection framework in Java?

Collection Framework is a combination of classes and interface, which is used to store and manipulate the data in the form of objects. It provides various classes such as ArrayList, Vector, Stack, and HashSet, etc. and interfaces such as List, Queue, Set, etc. for this purpose.

306. Explain various interfaces used in Collection framework?

Collection framework implements various interfaces, Collection interface and Map interface (java.util.Map) are the mainly used interfaces of Java Collection Framework. List of interfaces of Collection Framework is given below:

1. Collection interface: Collection (java.util.Collection) is the primary interface, and every collection must implement this interface.
2. List interface: List interface extends the Collection interface, and it is an ordered collection of objects. It contains duplicate elements. It also allows random access of elements.
3. Set interface: Set (java.util.Set) interface is a collection which cannot contain duplicate elements. It can only include inherited methods of Collection interface
4. Queue interface: Queue (java.util.Queue) interface defines queue data structure, which stores the elements in the form FIFO (first in first out).
5. Dequeue interface: it is a double-ended-queue. It allows the insertion and removal of elements from both ends. It implants the properties of both Stack and queue so it can perform LIFO (Last in first out) stack and FIFO (first in first out) queue, operations.
6. Map interface: A Map (java.util.Map) represents a key, value pair storage of elements. Map interface does not implement the Collection interface. It can only contain a unique key but can have duplicate elements. There are two interfaces which implement Map in java that are Map interface and Sorted Map.

307. What do you understand by BlockingQueue?

BlockingQueue is an interface which extends the Queue interface. It provides concurrency in the operations like retrieval, insertion, deletion. While retrieval of any element, it waits for the queue to be non-empty. While storing the elements, it waits for the available space. BlockingQueue cannot contain null elements, and implementation of BlockingQueue is thread-safe.

308. What is the advantage of properties file?

If you change the value in the properties file, you don't need to recompile the java class. So, it makes the application easy to manage. It is used to store information which is to be changed frequently.
Consider the following example.

```
import java.util.*;
import java.io.*;
public class Test {
public static void main(String[] args)throws Exception{
   FileReader reader=new FileReader("db.properties");

   Properties p=new Properties();
   p.load(reader);

   System.out.println(p.getProperty("user"));
   System.out.println(p.getProperty("password"));
}
}
```

Output

```
system
oracle
```

309. Why should we override equals() method?

The equals method is used to check whether two objects are the same or not. It needs to be overridden if we want to check the objects based on the property.

For example, Employee is a class that has 3 data members: id, name, and salary. However, we want to check the equality of employee object by the salary. Then, we need to override the equals() method.

310. How to synchronize List, Set and Map elements?

public static List synchronizedList(List l){}
public static Set synchronizedSet(Set s){}
public static SortedSet synchronizedSortedSet(SortedSet s){}
public static Map synchronizedMap(Map m){}
public static SortedMap synchronizedSortedMap(SortedMap m){}

311. What is the advantage of the generic collection?

1.There are three main advantages of using the generic collection.
2.if we use the generic class, we don't need typecasting.
3.It is type-safe and checked at compile time.
4.Generic confirms the stability of the code by making it bug detectable at compile time.

312. What is hash-collision in Hashtable and how it is handled in Java?

Two different keys with the same hash value are known as hash-collision. Two separate entries will be kept in a single hash bucket to avoid the collision. There are two ways to avoid hash-collision.
Separate Chaining
Open Addressing

313. What is the Dictionary class?

The Dictionary class provides the capability to store key-value pairs.

314. What is the default size of load factor in hashing based collection?

The default size of load factor is 0.75. The default capacity is computed as initial capacity * load factor. For example, 16 * 0.75 = 12. So, 12 is the default capacity of Map.

315. What do you understand by fail-fast?

The Iterator in java which immediately throws ConcurrentmodificationException, if any structural modification occurs in, is called as a Fail-fast iterator. Fail-fats iterator does not require any extra space in memory.

316. How to remove duplicates from ArrayList?

There are two ways to remove duplicates from the ArrayList.

1. **Using HashSet:** By using HashSet we can remove the duplicate element from the ArrayList, but it will not then preserve the insertion order.
2. **Using LinkedHashSet:** We can also maintain the insertion order by using LinkedHashSet instead of HashSet.

The Process to remove duplicate elements from ArrayList using the LinkedHashSet:

Copy all the elements of ArrayList to LinkedHashSet.

Empty the ArrayList using clear() method, which will remove all the elements from the list.

Now copy all the elements of LinkedHashset to ArrayList.

317. How to reverse ArrayList?

To reverse an ArrayList, we can use reverse() method of Collections class. Consider the following example.

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;
public class ReverseArrayList {
```

```
public static void main(String[] args) {
    List list = new ArrayList<>();
    list.add(10);
    list.add(50);
    list.add(30);
    Iterator i = list.iterator();
    System.out.println("printing the list. ..");
    while(i.hasNext())
    {
        System.out.println(i.next());
    }
    Iterator i2 = list.iterator();
    Collections.reverse(list);
    System.out.println("printing list in reverse order. ..");
    while(i2.hasNext())
    {
        System.out.println(i2.next());
    }
   }
}
```

Output:

```
printing the list....
10
50
30
printing list in reverse order....
30
50
10
```

318. How to sort ArrayList in descending order?

To sort the ArrayList in descending order, we can use the reverseOrder method of Collections class. Consider the following example.

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;
import java.util.List;

public class ReverseArrayList {
public static void main(String[] args) {
    List list = new ArrayList<>();
    list.add(10);
    list.add(50);
    list.add(30);
    list.add(60);
    list.add(20);
    list.add(90);

    Iterator i = list.iterator();
    System.out.println("printing the list. ..");
    while(i.hasNext())
    {
       System.out.println(i.next());
    }

   Comparator cmp = Collections.reverseOrder();
   Collections.sort(list,cmp);
    System.out.println("printing list in descending order. ..");
    Iterator i2 = list.iterator();
    while(i2.hasNext())
    {
```

```
        System.out.println(i2.next());
    }

}
}
```

Output:

```
printing the list....
10
50
30
60
20
90
printing list in descending order....
90
60
50
30
20
10
```

319. How to synchronize ArrayList?

We can synchronize ArrayList in two ways.

Using Collections.synchronizedList() method

Using CopyOnWriteArrayList<T>

320. When to use ArrayList and LinkedList?

LinkedLists are better to use for the update operations whereas ArrayLists are better to use for the search operations

321. Differentiate between

1.ArrayDeque and PriorityQueue

ArrayDeque	PriorityQueue
Internally uses array to store data	Internally usesminheap to store data
Order of insertion is preserved	Order of insertion is not preserved
Highest priority element is not present at the beginning of the queue	Highest priority element is present at the beginning of the queue
Iterators as well as descending iterators can be used	Only iterators can be uses

2.Vector and ArrayList

Vector	ArrayList
Introduced from java 1.2	Part of collections from the begining
Vector is a legacy class	It is not a legacy class
Vectors are synchronized	ArrayListsare not synchronized
Thread safe	Not thread safe
Performance is relatively low	Performance is relatively high
FailFast error does not occur	It is failfast

3.ArrayList and Linked List

Vector	ArrayList
Introduced from java 1.2	Part of collections from the begining
Vector is a legacy class	It is not a legacy class

Vector is synchronized	ArrayList is not synchronized
Thread safe	Not thread safe
Performance is relatively low	Performance is relatively high
Fail-Fast error does not occur	It is fail-fast

4.Array and ArrayList

Array	**ArrayList**
Holds primitive data	Can store objects
Is not dynamic in nature	Dynamic in nature
Can store homogeneous data	Can store heterogeneous data
Arrays does not provide methods	Provides methods to perform various operations
Contents can be accepted using loops	Can accept using loops, iterator or list iterator
Present from beginning of java	Introduced in java 1.2
Not part of collection	Part of collection

5.TreeSet and HashSet

TreeSet	**HashSet**
Internally makes use of balanced binary search tree	Internally makes use of hashing
Objects are stored in sorted order	Objects are not stored in sorted order
The time complexity for each operation is O(log2n)	Time complexity of each operation is O(1)

TreeSet has implemented from navigable set	HashSet has been extended from Abstract set
Descending Iterator can be applied on tree set	Descending Iterator cannot be applied

6.HashSet and LinkedHashSet

HashSet	**LinkedHashSet**
Order of insertion not preserved	Order of insertion preserved
Extends from abstract set	Extends from HashSet

7.Comparable and Comparator

Comparable	**Comparator**
Expects the implementation of compareTo()	Expectsthe implementation of compare()
General format of compareTo() public intcompareTo(Object y)	General format of compare() public intcompare(Object y,Object x)
This is used to refer to first object	This is not used to refer to first object
Comparable interface is useful if and only if the source code of the class is available and open for modification	Comparator interface is useful even if the source code of the class is not available and not open for modification

8.Collection and Collections

Collection	**Collections**
Collection is an interface which can be implemented the List, Set,Sorted Set,and Queue	Collections is an utility class which contain the utility methods such as sorting etc.

9.Iterator and Enumeration

Iterator	Enumeration
Iterator allows to remove elements from collection during traversal	Enumeration doesn't allow
More secure and safe	Less secure and safe

10. Iterator and ListIterator

Iterator	ListIterator
Iterator is used for traversing List and Set both.	Used to traverse List only, we cannot traverse Set using ListIterator.
We can traverse in only forward direction using Iterator	Using ListIterator, we can traverse a List in both the directions (forward and Backward).
We cannot obtain indexes while using Iterator	We can obtain indexes at any point of time while traversing a list using ListIterator. The methods nextIndex() and previousIndex() are used for this purpose.
We cannot add element to collection while traversing it using Iterator, it throws ConcurrentModificationException when you try to do it.	We can add element at any point of time while traversing a list using ListIterator.
We cannot replace the existing element value when using Iterator.	By using set(E e) method of ListIterator we can replace the last element returned by next() or previous() methods.

11. List and Set

List	Set
List is an ordered collection it maintains the insertion order, which means upon displaying the list content it will display the elements in the same order in which they got inserted into the list.	Set is an unordered collection, it doesn't maintain any order. There are few implementations of Set which maintains the order such as LinkedHashSet (It maintains the elements in insertion order).
List allows duplicates	Set doesn't allow duplicate elements. All the elements of a Set should be unique if you try to insert the duplicate element in Set it would replace the existing value.
List implementations: ArrayList, LinkedList etc.	Set implementations: HashSet,LinkedHashSet ,TreeSet etc.
List allows any number of null values.	Set can have only a single null value at most.

12. Set and Map

Set	Map
Set doesn't allow duplicates. Set and all of the classes which implements Set interface should have unique elements.	Map stores the elements as key & value pair. Map doesn't allow duplicate keys while it allows duplicate values.
Set allows single null value at most.	Map can have single null key at most and any number of null values.

13. HashSet and HashMap

HashSet	HashMap
HashSet class implements the Set interface	HashMap class implements the Map interface
In HashSet we store objects(elements or values)	HashMap is used for storing key & value pairs
HashSet permits to have a single null value.	HashMap permits single null key and any number of null values.
HashSet does not allow duplicate elements that means you can not store duplicate values in HashSet.	HashMap does not allow duplicate keys however it allows to have duplicate values.

14. HashMap and TreeMap

HashMap	TreeMap
HashMap returns unordered values	TreeMap returns the elements in ascending order
The performance of HashMap is higher	Performance on the lower side

15. HashMap and HashTable

HashMap	HashTable
HashMap is non synchronized. It is not-thread safe and can't be shared between many threads without proper synchronization code.	Hashtable is synchronized. It is thread-safe and can be shared with many threads.
HashMap allows one null key and multiple null values.	Hashtable doesn't allow any null key or value.
HashMap is a new class introduced in JDK 1.2.	Hashtable is a legacy class.
HashMap is fast.	Hashtable is slow.

We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap);	Hashtable is internally synchronized and can't be unsynchronized.
HashMap is traversed by Iterator.	Hashtable is traversed by Enumerator and Iterator.
Iterator in HashMap is fail-fast.	Enumerator in Hashtable is not fail-fast.
HashMap inherits AbstractMap class.	Hashtable inherits Dictionary class.

321. What is an Iterator ?

The Iterator interface provides a number of methods that are able to iterate over any Collection. Each Java Collection contains the Iterator method that returns an Iterator instance. Iterators are capable of removing elements from the underlying collection during the iteration.

322. What is difference between fail-fast and fail-safe ?

The Iterator's fail-safe property works with the clone of the underlying collection and thus, it is not affected by any modification in the collection. All the collection classes in java.util package are fail-fast, while the collection classes in java.util.concurrent are fail-safe. Fail-fast iterators throw a ConcurrentModificationException, while fail-safe iterator never throws such an exception.

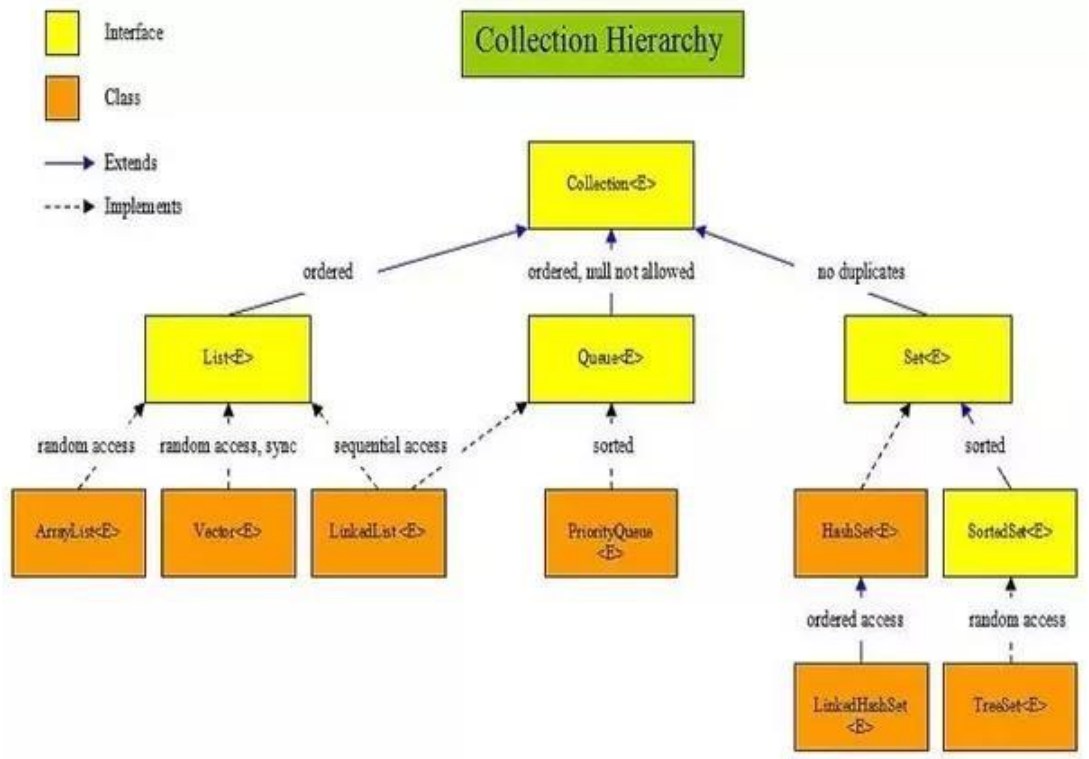
323.How HashMap works in Java ?

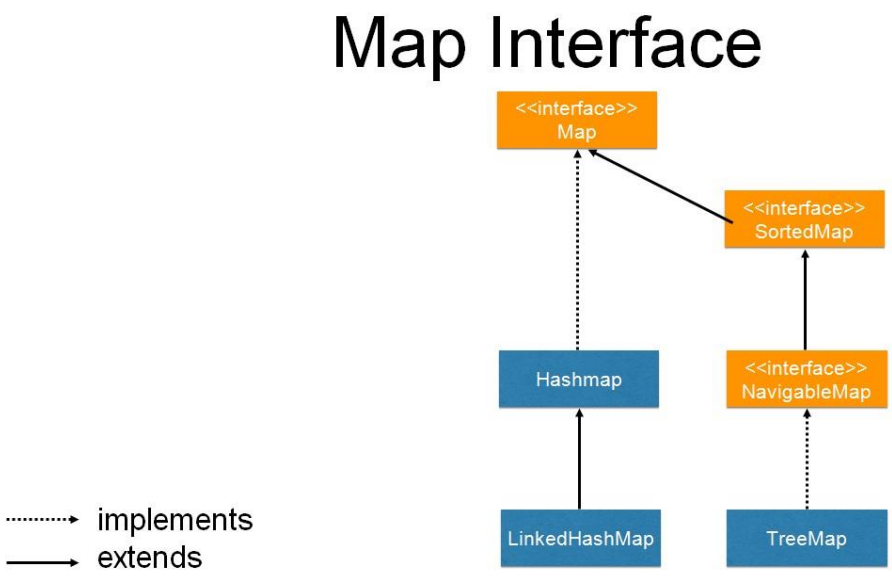
A HashMap in Java stores key-value pairs. The HashMap requires a hash function and uses hashCode and equals methods, in order to put and retrieve elements to and from the collection respectively. When the put method is invoked, the HashMap calculates the hash value of the key and stores the pair in the appropriate index inside the collection. If the key exists, its value is updated with the new value. Some important characteristics of a HashMap are its capacity, its load factor and the threshold resizing.

324. What is Comparable and Comparator interface?

List their differences. Java provides the Comparable interface, which contains only one method, called compareTo(). This method compares two objects, in order to impose an order between them. Specifically, it returns a negative integer, zero, or a positive integer to indicate that the input object is less than, equal or greater than the existing object. Java provides the Comparator interface, which contains two methods, called compare() and equals). The first method compares its two input arguments and imposes an order between them. It returns a negative integer, zero, or a positive integer to indicate that the first argument is less than, equal to, or greater than the second. The second method requires an object as a parameter and aims to decide whether the input object is equal to the comparator. The method returns true, only if the specified object is also a comparator and it imposes the same ordering as the comparator.

325.Write Collection Hierarchy





326. What happens when you compile and run the below program?

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Quiz38 {

    public static void main(String[] args) {

        List<Integer> list = new ArrayList<Integer>();

        Integer[] arr = {2,10,3};

        list = Arrays.asList(arr);

        list.set(0, 3);

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

```
        list.add(1);

        System.out.println(list);
    }

}
```

Answer:

Output :

[3,10,3], followed by exception

This is because Arrays.asList() returns a fixed-size list backed by the specified array. Therefore, the ArrayList can't grow.

So, when add() is called, an exception is thrown.

327. What will be output of following program?

```
import java.util.ArrayList;
import java.util.List;

public class ArrayListDemo {

    public static void main(String[] args) {

        List list = new ArrayList();

        list.add(10);
        list.add(10);

        System.out.print(list.size());

        list.remove(new Integer(10));

        System.out.print(list.size());
    }
}
```

Answer:
Output:
21
This is because ArralyList can contain duplicate elements.
ArrayList remove() method only removes the first occurrence of a matching element.

328. What will be the output of following Java code

```
import java.util.PriorityQueue;

public class PriorityQueueExample {

   public static void main(String[] args) {
      PriorityQueue<String> pQueue = new PriorityQueue<String>();
      pQueue.add("Apple");
      pQueue.add("Nokia");
      pQueue.add("Samsung");
      pQueue.add("Apple");

      System.out.print(pQueue.poll() + " " + pQueue.poll());
      System.out.print(" " + pQueue.peek() + " " + pQueue.poll());
   }

}
```

Answer:
Output:
“Apple Apple Nokia Nokia”
Note:
PriorityQueue keeps elements sorted and it can have duplicates.
add() and offer() methods both offer same functionality.
poll() method removes the first element in queue and returns it, while peek() method returns the first element without removing it.

329. What will be the result for below program?

```
package com.Kodnest;

public class Student {

   int rollNumber;

   Student(int n){
      rollNumber = n;
   }
}

package com.Kodnest;

import java.util.HashSet;
import java.util.Set;

public class HashSetDemo {

   public static void main(String[] args) {

      Set<Student> students = new HashSet<Student>();

      students.add(new Student(1));
      students.add(new  Student(3));
      students.add(new Student(4));
      students.add(new  Student(1));
      students.add(new Student(3));

      System.out.println(students.size());
   }

}
```

Answer:
Output
5

Because Student doesn't override equals(), there are 5 objects in the HashSet.

330. Predict output of following program :

```
package com.Kodnest;

public class Employee implements Comparable<Employee>{

    int id;

    String name;

   Employee(int id, String name){

       this.id = id;

       this.name = name;

   }

   @Override

   public int compareTo(Employee emp) {

       return this.name.compareTo(emp.name);

   }

}

import java.util.Comparator;


public class EmployeeComparator implements Comparator<Employee>{

   @Override

   public int compare(Employee emp1, Employee emp2)
```

```
{
        return emp2.id - emp1.id;
    }
}
import java.util.TreeSet;

public class TreeSetDemo{

   public static void main(String[] args) {

      TreeSet<Employee> empTreeSet = new TreeSet<Employee>(new
EmployeeComparator());

      Employee emp1 = new Employee(20, "Clark");
      Employee emp2 = new Employee(24, "Bernie");
      Employee emp3 = new Employee(3, "Alex");
      empTreeSet.add(emp1);
      empTreeSet.add(emp2);
      empTreeSet.add(emp3);

      for(Employee emp : empTreeSet)
            System.out.print(emp.name + " ");

   }
}
```

Answer:

Output :

Bernie Clark Alex

331. How to copy Set content to another HashSet?

```
import java.util.HashSet;

public class MyHashSetCopy
{
   public static void main(String a[])
{
       HashSet<String> hs = new HashSet<String>();
       //add elements to HashSet
       hs.add("first");
       hs.add("second");
       hs.add("third");
       System.out.println(hs);
       HashSet<String> subSet = new HashSet<String>();
       subSet.add("s1");
       subSet.add("s2");
       hs.addAll(subSet);
       System.out.println("HashSet content after adding another
collection:");
       System.out.println(hs);
   }
}
```

Output:

```
[second, third, first]
HashSet content after adding another collection:
[s2, s1, second, third, first]
```

332. How to copy all elements from HashSet to an array?

```
import java.util.HashSet;
public class MyHashSetToArray
 {
   public static void main(String a[])
        {
       HashSet<String> hs = new HashSet<String>();
```

```
        //add elements to HashSet
        hs.add("first");
        hs.add("second");
        hs.add("third");
        System.out.println("HashSet content: ");
        System.out.println(hs);
        String[] strArr = new String[hs.size()];
        hs.toArray(strArr);
        System.out.println("Copied array content:");
        for(String str:strArr){
            System.out.println(str);
        }
    }
}
```

Output:
HashSet content:
[second, third, first]
Copied array content:
second
third
first

333. How to eliminate duplicate user defined objects from HashSet?

```
import  java.util.HashSet;
public class MyDistElementEx {
    public static void main(String a[]){
        HashSet<Price> lhm = new HashSet<Price>();
        lhm.add(new Price("Banana", 20));
        lhm.add(new Price("Apple", 40));
        lhm.add(new Price("Orange", 30));
        for(Price pr:lhm){
            System.out.println(pr);
```

```
        }
        Price duplicate = new Price("Banana", 20);
        System.out.println("inserting duplicate object...");
        lhm.add(duplicate);
        System.out.println("After insertion:");
        for(Price pr:lhm){
            System.out.println(pr);
        }
    }
}

class Price{
    private String item;
    private int price;

    public Price(String itm, int pr){
        this.item = itm;
        this.price = pr;
    }

    public int hashCode(){
        System.out.println("In hashcode");
        int hashcode = 0;
        hashcode = price*20;
        hashcode += item.hashCode();
        return hashcode;
    }

    public boolean equals(Object obj){
        System.out.println("In equals");
        if (obj instanceof Price) {
            Price pp = (Price) obj;
            return (pp.item.equals(this.item) && pp.price == this.price);
        } else {
```

```
            return false;
        }
    }

    public String getItem() {
        return item;
    }
    public void setItem(String item) {
        this.item = item;
    }
    public int getPrice() {
        return price;
    }
    public void setPrice(int price) {
        this.price = price;
    }

    public String toString(){
        return "item: "+item+"  price: "+price;
    }
}
```

Output:

```
In hashcode
In hashcode
In hashcode
item: Apple price: 40
item: Orange price: 30
item: Banana  price: 20
inserting duplicate object...
In hashcode
In equals
After insertion:
item: Apple  price: 40
```

item: Orange price: 30
item: Banana price: 20

334. How to sort LinkedList using Comparator?

```
import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedList;

public class MyLinkedListSort {

   public static void main(String a[]){

      LinkedList<Empl> list = new LinkedList<Empl>();
      list.add(new Empl("Ram",3000));
      list.add(new Empl("John",6000));
      list.add(new Empl("Crish",2000));
      list.add(new Empl("Tom",2400));
      Collections.sort(list,new MySalaryComp());
      System.out.println("Sorted list entries: ");
      for(Empl e:list){
         System.out.println(e);
      }
   }
}

class MySalaryComp implements Comparator<Empl>{

   @Override
   public int compare(Empl e1, Empl e2) {
      if(e1.getSalary() < e2.getSalary()){
         return 1;
      } else {
         return -1;
      }
   }
```

```
}

class Empl{

   private String name;
   private int salary;

   public Empl(String n, int s){
      this.name = n;
      this.salary = s;
   }

   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public int getSalary() {
      return salary;
   }
   public void setSalary(int salary) {
      this.salary = salary;
   }
   public String toString(){
      return "Name: "+this.name+"-- Salary: "+this.salary;
   }
}
```

Output:

```
Sorted list entries:
Name: John-- Salary: 6000
Name: Ram-- Salary: 3000
```

Name: Tom-- Salary: 2400
Name: Crish-- Salary: 2000

335. How to reverse LinkedList content?

```
import java.util.Collections;
import java.util.LinkedList;

public class MyLinkedListReverse {

   public static void main(String a[]){

      LinkedList<String> list = new LinkedList<String>();
      list.add("Java");
      list.add("Cric");
      list.add("Play");
      list.add("Watch");
      list.add("Glass");
      Collections.reverse(list);
      System.out.println("Results after reverse operation:");
      for(String str: list){
         System.out.println(str);
      }
   }
}
```

Output:

```
Results after reverse operation:
Glass
Watch
Play
Cric
Java
```

336. How to shuffle elements in LinkedList?

```
import java.util.Collections;
import java.util.LinkedList;
```

```
public class MyLinkedListShuffle {
    public static void main(String a[]){

        LinkedList<String> list = new LinkedList<String>();
        list.add("Java");
        list.add("Cric");
        list.add("Play");
        list.add("Watch");
        list.add("Glass");
        list.add("Movie");
        list.add("Girl");

        Collections.shuffle(list);
        System.out.println("Results after shuffle operation:");
        for(String str: list){
            System.out.println(str);
        }

        Collections.shuffle(list);
        System.out.println("Results after shuffle operation:");
        for(String str: list){
            System.out.println(str);
        }
    }
}
```

Output:

```
Results after shuffle operation:
Movie
Girl
Watch
Glass
```

```
Java
Cric
Play
Results after shuffle operation:
Glass
Watch
Play
Girl
Cric
Movie
Java
```

337. How to convert list to csv string format?

```
import java.util.LinkedList;
import java.util.List;

public class MyListToCsvString {

   public String getListAsCsvString(List<String> list){

      StringBuilder sb = new StringBuilder();
      for(String str:list){
         if(sb.length() != 0){
            sb.append(",");
         }
         sb.append(str);
      }
      return sb.toString();
   }

   public static void main(String a[]){

      List<String> li1 = new LinkedList<String>(){
         {
```

```
            this.add("animal");
            this.add("nuts");
            this.add("java");
        }
    };
    MyListToCsvString mtc = new MyListToCsvString();
    System.out.println(mtc.getListAsCsvString(li1));
    List<String> li2 = new LinkedList<String>(){
        {
            this.add("java");
            this.add("unix");
            this.add("c++");
        }
    };
    System.out.println(mtc.getListAsCsvString(li2));
  }
}
```

Output:

animal,nuts,java

java,unix,c++

338. How to remove elements from LinkedList?

Note:

Below example shows how to remove or delete an element from LinkedList. LinkedList provides few methods to remove elements, those methods are:

remove(): Retrieves and removes the head (first element) of this list.

remove(index): Removes the element at the specified position in this list.

remove(object): Removes the first occurrence of the specified element from this list, if it is present.

removeFirst(): Removes and returns the first element from this list.

removeFirstOccurrence(object): Removes the first occurrence of the specified element in this list (when traversing the list from head to tail).

removeLast(): Removes and returns the last element from this list.

removeLastOccurrence(object): Removes the last occurrence of the specified element in this list (when traversing the list from head to tail).

```
public class MyAllRemoveOprs
 {
   public static void main(String a[])
{
      LinkedList<String> arrl = new LinkedList<String>();
      arrl.add("First");
      arrl.add("Second");
      arrl.add("Third");
      arrl.add("Random");
      arrl.add("four");
      arrl.add("five");
      arrl.add("six");
      arrl.add("seven");
      arrl.add("eight");
      arrl.add("nine");
      System.out.println(arrl);
      System.out.println("Remov() method:"+arrl.remove());
      System.out.println("After remove() method call:");
      System.out.println(arrl);
      System.out.println("remove(index) method:"+arrl.remove(2));
      System.out.println("After remove(index) method call:");
      System.out.println(arrl);
      System.out.println("Remov(object) method:"+arrl.remove("six"));
      System.out.println("After remove(object) method call:");
      System.out.println(arrl);
      System.out.println("removeFirst() method:"+arrl.removeFirst());
      System.out.println("After removeFirst() method call:");
      System.out.println(arrl);
      System.out.println("removeFirstOccurrence() method:"
                  +arrl.removeFirstOccurrence("eight"));
      System.out.println("After removeFirstOccurrence() method call:");
```

```
        System.out.println(arrl);
        System.out.println("removeLast() method:"+arrl.removeLast());
        System.out.println("After removeLast() method call:");
        System.out.println(arrl);
        System.out.println("removeLastOccurrence() method:"
                        +arrl.removeLastOccurrence("five"));
        System.out.println("After removeLastOccurrence() method call:");
        System.out.println(arrl);
    }
}
```

Output:

```
[First, Second, Third, Random, four, five, six, seven, eight, nine]
Remov() method:First
After remove() method call:
[Second, Third, Random, four, five, six, seven, eight, nine]
remove(index) method:Random
After remove(index) method call:
[Second, Third, four, five, six, seven, eight, nine]
Remov(object) method:true
After remove(object) method call:
[Second, Third, four, five, seven, eight, nine]
removeFirst() method:Second
After removeFirst() method call:
[Third, four, five, seven, eight, nine]
removeFirstOccurrence() method:true
After removeFirstOccurrence() method call:
[Third, four, five, seven, nine]
removeLast()  method:nine
After removeLast() method call:
[Third, four, five, seven]
removeLastOccurrence()  method:true
After removeLastOccurrence() method call:
[Third, four, seven]
```

339. How to compare two LinkedHashSet and retain elements which are same on both LinkedHashSet?

```
import java.util.LinkedHashSet;

public class MyLhsRetainEx {

   public static void main(String a[]){

      LinkedHashSet<String> lhs = new LinkedHashSet<String>();
      //add elements to LinkedHashSet
      lhs.add("first");
      lhs.add("second");
      lhs.add("third");
      lhs.add("apple");
      lhs.add("rat");
      System.out.println(lhs);
      LinkedHashSet<String> subSet = new LinkedHashSet<String>();
      subSet.add("rat");
      subSet.add("second");
      subSet.add("first");
      lhs.retainAll(subSet);
      System.out.println("LinkedHashSet content:");
      System.out.println(lhs);
   }
}
```

Output:

```
[first, second, third, apple, rat]
LinkedHashSet content:
[first, second, rat]
```

340. How to create a TreeSet with comparator?

```
import java.util.Comparator;
import java.util.TreeSet;
```

```
public class MySetWithCompr
{
   public static void main(String a[])
{
      TreeSet<String> ts = new TreeSet<String>(new MyComp());
      ts.add("RED");
      ts.add("ORANGE");
      ts.add("BLUE");
      ts.add("GREEN");
      System.out.println(ts);
   }
}

class MyComp implements Comparator<String>
{
   @Override
   public int compare(String str1, String str2)
 {
      return str1.compareTo(str2);
   }
}
```

Output:
[BLUE, GREEN, ORANGE, RED]

341. How to get subset from sorted set?

```
import java.util.Comparator;
import java.util.Set;
import java.util.TreeSet;

public class MySetSublist {
   public static void main(String a[]){
      TreeSet<String> ts = new TreeSet<String>(new MyStrComp());
      ts.add("RED");
```

```
        ts.add("ORANGE");
        ts.add("BLUE");
        ts.add("GREEN");
        ts.add("WHITE");
        ts.add("BROWN");
        ts.add("YELLOW");
        ts.add("BLACK");
        System.out.println(ts);
        Set<String> subSet = ts.subSet("GREEN", "WHITE");
        System.out.println("sub set: "+subSet);
        subSet = ts.subSet("GREEN", true, "WHITE", true);
        System.out.println("sub set: "+subSet);
        subSet = ts.subSet("GREEN", false, "WHITE", true);
        System.out.println("sub set: "+subSet);
    }
}

class MyStrComp implements Comparator<String>{

    @Override
    public int compare(String str1, String str2) {
        return str1.compareTo(str2);
    }

}
```

Output:

```
[BLACK, BLUE, BROWN, GREEN, ORANGE, RED, WHITE, YELLOW]
sub set: [GREEN, ORANGE, RED]
sub set: [GREEN, ORANGE, RED, WHITE]
sub set: [ORANGE, RED, WHITE]
```

342. How to avoid duplicate user defined objects in TreeSet?

```
import java.util.Comparator;
import java.util.Set;
```

```
import java.util.TreeSet;

public class MyUserDuplicates {

   public static void main(String a[]){

      Set<Emp> ts = new TreeSet<Emp>(new EmpComp());
      ts.add(new Emp(201,"John",40000));
      ts.add(new Emp(302,"Krish",44500));
      ts.add(new Emp(146,"Tom",20000));
      ts.add(new Emp(543,"Abdul",10000));
      ts.add(new Emp(12,"Dinesh",50000));
      //adding duplicate entry
      ts.add(new Emp(146,"Tom",20000));
      //check duplicate entry is there or not
      for(Emp e:ts){
         System.out.println(e);
      }
   }
}

class EmpComp implements Comparator<Emp>{

   @Override
   public int compare(Emp e1, Emp e2) {
      if(e1.getEmpId() == e2.getEmpId()){
         return 0;
      } if(e1.getEmpId() < e2.getEmpId()){
         return 1;
      } else {
         return -1;
      }
   }
}
```

```
class Emp {

   private int empId;
   private String empName;
   private int empSal;

   public Emp(int id, String name, int sal){
      this.empId = id;
      this.empName = name;
      this.empSal = sal;
   }

   public int getEmpId() {
      return empId;
   }

   public void setEmpId(int empId) {
      this.empId = empId;
   }

   public String getEmpName() {
      return empName;
   }

   public void setEmpName(String empName) {
      this.empName = empName;
   }

   public int getEmpSal() {
      return empSal;
   }
   public void setEmpSal(int empSal) {
      this.empSal = empSal;
```

```
    }

    public String toString(){
        return empId+" : "+empName+" : "+empSal;
    }
}
```

Output:

```
543 : Abdul : 10000
302 : Krish : 44500
201 : John : 40000
146 : Tom : 20000
12 : Dinesh : 50000
```

343. How to copy HashMap content to another HashMap?

```
import java.util.HashMap;

public class MyHashMapCopy {

    public static void main(String a[]){
        HashMap<String, String> hm = new HashMap<String, String>();
        //add key-value pair to hashmap
        hm.put("first", "FIRST INSERTED");
        hm.put("second", "SECOND INSERTED");
        hm.put("third","THIRD INSERTED");
        System.out.println(hm);
        HashMap<String, String> subMap = new HashMap<String, String>();
        subMap.put("s1", "S1 VALUE");
        subMap.put("s2", "S2 VALUE");
        hm.putAll(subMap);
        System.out.println(hm);
    }
}
```

Output:

{second=SECOND INSERTED, third=THIRD INSERTED, first=FIRST INSERTED}
{s2=S2 VALUE, s1=S1 VALUE, second=SECOND INSERTED, third=THIRD INSERTED, first=FIRST INSERTED}

344. How to eliminate duplicate user defined objects as a key from HashMap?

```
import java.util.HashMap;
import java.util.Set;

public class MyDuplicateKeyEx {

   public static void main(String a[]){

      HashMap<Price, String> hm = new HashMap<Price, String>();
      hm.put(new Price("Banana", 20), "Banana");
      hm.put(new Price("Apple", 40), "Apple");
      hm.put(new Price("Orange", 30), "Orange");
      printMap(hm);
      Price key = new Price("Banana", 20);
      System.out.println("Adding duplicate key...");
      hm.put(key, "Grape");
      System.out.println("After adding dulicate key:");
      printMap(hm);
   }

   public static void printMap(HashMap<Price, String> map){

      Set<Price> keys = map.keySet();
      for(Price p:keys){
         System.out.println(p+"==>"+map.get(p));
      }
   }
}
```

```
class Price{

   private String item;
   private int price;

   public Price(String itm, int pr){
      this.item = itm;
      this.price = pr;
   }

   public int hashCode(){
      int hashcode = 0;
      hashcode = price*20;
      hashcode += item.hashCode();
      return hashcode;
   }

   public boolean equals(Object obj){
      if (obj instanceof Price) {
         Price pp = (Price) obj;
         return (pp.item.equals(this.item) && pp.price == this.price);
      } else {
         return false;
      }
   }

   public String getItem() {
      return item;
   }
   public void setItem(String item) {
      this.item = item;
   }
   public int getPrice() {
      return price;
```

```
    }
    public void setPrice(int price) {
        this.price = price;
    }

    public String toString(){
        return "item: "+item+"  price: "+price;
    }
}
```

Output:

```
item: Apple price: 40==>Apple
item: Orange price: 30==>Orange
item: Banana price: 20==>Banana
Adding duplicate key...
After adding dulicate key:
item: Apple price: 40==>Apple
item: Orange price: 30==>Orange
item: Banana  price: 20==>Grape
```

345. How to eliminate duplicate user defined objects as a key from HashMap?

```
import java.util.HashMap;
import java.util.Set;

public class MyDuplicateKeyEx {

    public static void main(String a[]){

        HashMap<Price, String> hm = new HashMap<Price, String>();
        hm.put(new Price("Banana", 20), "Banana");
        hm.put(new Price("Apple", 40), "Apple");
        hm.put(new Price("Orange", 30), "Orange");
        printMap(hm);
        Price key = new Price("Banana", 20);
```

```
        System.out.println("Adding duplicate key...");
        hm.put(key, "Grape");
        System.out.println("After adding dulicate key:");
        printMap(hm);
    }

    public static void printMap(HashMap<Price, String> map){

        Set<Price> keys = map.keySet();
        for(Price p:keys){
            System.out.println(p+"==>"+map.get(p));
        }
    }
}

class Price{

    private String item;
    private int price;

    public Price(String itm, int pr){
        this.item = itm;
        this.price = pr;
    }

    public int hashCode(){
        int hashcode = 0;
        hashcode = price*20;
        hashcode += item.hashCode();
        return hashcode;
    }

    public boolean equals(Object obj){
        if (obj instanceof Price) {
```

```
            Price pp = (Price) obj;
            return (pp.item.equals(this.item) && pp.price == this.price);
        } else {
            return false;
        }
    }

    public String getItem() {
        return item;
    }
    public void setItem(String item) {
        this.item = item;
    }
    public int getPrice() {
        return price;
    }
    public void setPrice(int price) {
        this.price = price;
    }

    public String toString(){
        return "item: "+item+"  price: "+price;
    }
}
```

Output:

```
item: Apple price: 40==>Apple
item: Orange price: 30==>Orange
item: Banana price: 20==>Banana
Adding duplicate key...
After adding dulicate key:
item: Apple  price: 40==>Apple
item: Orange price: 30==>Orange
item: Banana  price: 20==>Grape
```

346. How to sort keys in TreeMap by using Comparator?

```
import java.util.Comparator;
import java.util.TreeMap;

public class MyTreeMapComparator {

   public static void main(String a[]){
      //the treemap sorts by key
      TreeMap<String, String> hm = new TreeMap<String, String>(new
MyComp());
      //add key-value pair to TreeMap
      hm.put("java", "language");
      hm.put("computer", "machine");
      hm.put("india","country");
      hm.put("mango","fruit");
      System.out.println(hm);
   }
}

class MyComp implements Comparator<String>{

   @Override
   public int compare(String str1, String str2) {
      return str1.compareTo(str2);
   }

}
```

Output:

```
{computer=machine, india=country, java=language, mango=fruit}
```

347. How to sort keys in TreeMap by using Comparator with user define objects?

```
import java.util.Comparator;
import java.util.Set;
import java.util.TreeMap;

public class MyTMCompUserDefine {

   public static void main(String a[]){
      //By using name comparator (String comparison)
      TreeMap<Empl,String> tm = new TreeMap<Empl, String>(new
MyNameComp());
      tm.put(new Empl("Ram",3000), "RAM");
      tm.put(new Empl("John",6000), "JOHN");
      tm.put(new Empl("Crish",2000), "CRISH");
      tm.put(new Empl("Tom",2400), "TOM");
      Set<Empl> keys = tm.keySet();
      for(Empl key:keys){
         System.out.println(key+" ==> "+tm.get(key));
      }
      System.out.println("====================================");
      //By using salary comparator (int comparison)
      TreeMap<Empl,String> trmap = new TreeMap<Empl, String>(new
MySalaryComp());
      trmap.put(new Empl("Ram",3000), "RAM");
      trmap.put(new Empl("John",6000), "JOHN");
      trmap.put(new Empl("Crish",2000), "CRISH");
      trmap.put(new Empl("Tom",2400), "TOM");
      Set<Empl> ks = trmap.keySet();
      for(Empl key:ks){
         System.out.println(key+" ==> "+trmap.get(key));
      }
   }
}
```

```
class MyNameComp implements Comparator<Empl>{

   @Override
   public int compare(Empl e1, Empl e2) {
      return e1.getName().compareTo(e2.getName());
   }
}

class MySalaryComp implements Comparator<Empl>{

   @Override
   public int compare(Empl e1, Empl e2) {
      if(e1.getSalary() > e2.getSalary()){
         return 1;
      } else {
         return -1;
      }
   }
}

class Empl{

   private String name;
   private int salary;

   public Empl(String n, int s){
      this.name = n;
      this.salary = s;
   }

   public String getName() {
      return name;
   }
```

```
    public void setName(String name) {
        this.name = name;
    }
    public int getSalary() {
        return salary;
    }
    public void setSalary(int salary) {
        this.salary = salary;
    }
    public String toString(){
        return "Name: "+this.name+"-- Salary: "+this.salary;
    }
}
```

Output:

```
Name: Crish-- Salary: 2000 ==> CRISH
Name: John-- Salary: 6000 ==> JOHN
Name: Ram-- Salary: 3000 ==> RAM
Name: Tom-- Salary: 2400 ==> TOM
==================================
Name: Crish-- Salary: 2000 ==> CRISH
Name: Tom-- Salary: 2400 ==> TOM
Name: Ram-- Salary: 3000 ==> RAM
Name: John-- Salary: 6000 ==> JOHN
```

348. How to reverse sorted keys in a TreeMap?

```
import java.util.Comparator;
import java.util.Map;
import java.util.TreeMap;

public class MyReverseOrderMap {

    public static void main(String a[]){
        //the treemap sorts by key
```

```
        TreeMap<String, String> hm = new TreeMap<String, String>(new
MyCopr());
        //add key-value pair to TreeMap
        hm.put("java", "language");
        hm.put("computer", "machine");
        hm.put("india","country");
        hm.put("mango","fruit");
        hm.put("game","cricket");
        System.out.println("TreeMap Entries:");
        System.out.println(hm);
        Map<String, String> rm = hm.descendingMap();
        System.out.println("Reverse Map Content: ");
        System.out.println(rm);
    }
}

class MyCopr implements Comparator<String>{

    @Override
    public int compare(String str1, String str2) {
        return str1.compareTo(str2);
    }

}
```

Output:

```
TreeMap Entries:
{computer=machine, game=cricket, india=country, java=language,
mango=fruit}
Reverse Map Content:
{mango=fruit, java=language, india=country, game=cricket,
computer=machine}
```

349. How to search user defined object from a List by using binary search and by using comparator?

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class MyListBinarySearch {

   public static void main(String a[]){

      List<Emp> empList = new ArrayList<Emp>();
      empList.add(new Emp(12,"Dinesh",50000));
      empList.add(new Emp(146,"Tom",20000));
      empList.add(new Emp(201,"John",40000));
      empList.add(new Emp(302,"Krish",44500));
      empList.add(new Emp(543,"Abdul",10000));

      Emp searchKey = new Emp(201,"John",40000);
      int index = Collections.binarySearch(empList, searchKey, new
EmpComp());
      System.out.println("Index of the searched key: "+index);
   }
}

class EmpComp implements Comparator<Emp>{

   public int compare(Emp e1, Emp e2) {
      if(e1.getEmpId() == e2.getEmpId()){
         return 0;
      } else {
         return -1;
      }
   }
```

```
}

class Emp {

   private int empId;
   private String empName;
   private int empSal;

   public Emp(int id, String name, int sal){
      this.empId = id;
      this.empName = name;
      this.empSal = sal;
   }

   public int getEmpId() {
      return empId;
   }

   public void setEmpId(int empId) {
      this.empId = empId;
   }

   public String getEmpName() {
      return empName;
   }

   public void setEmpName(String empName) {
      this.empName = empName;
   }

   public int getEmpSal() {
      return empSal;
   }
   public void setEmpSal(int empSal) {
```

```
        this.empSal = empSal;
    }

    public String toString(){
        return empId+" : "+empName+" : "+empSal;
    }
}
```

Output:

Index of the searched key: 2

350. Write an example for Collections.checkedList() method.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class MyCheckedList {

    public static void main(String a[]){

        List myList = new ArrayList();
        myList.add("one");
        myList.add("two");
        myList.add("three");
        myList.add("four");
        List chkList = Collections.checkedList(myList, String.class);
        System.out.println("Checked list content: "+chkList);
        //you can add any type of elements to myList object
        myList.add(10);
        //you cannot add any type of elements to chkList object, doing so
        //throws  ClassCastException
        chkList.add(10); //throws ClassCastException
    }
}
```

Output:

```
Checked list content: [one, two, three, four]
Exception in thread "main" java.lang.ClassCastException: Attempt to
insert class java.lang.Integer element into
                                    collection with element type class
java.lang.String
        at
java.util.Collections$CheckedCollection.typeCheck(Collections.java:2202)
        at
java.util.Collections$CheckedCollection.add(Collections.java:2243)
        at
com.java2novice.collections.MyCheckedList.main(MyCheckedList.java:22)
```

351. Write an example for Collections.checkedSet() method.

```
import java.util.Collections;
import java.util.HashSet;
import java.util.Set;

public class MyCheckedSet {

   public static void main(String a[]){

      Set mySet = new HashSet();
      mySet.add("one");
      mySet.add("two");
      mySet.add("three");
      mySet.add("four");
      Set chkSet = Collections.checkedSet(mySet, String.class);
      System.out.println("Checked set content: "+chkSet);
      //you can add any type of elements to mySet object
      mySet.add(10);
      //you cannot add any type of elements to chkSet object, doing so
      //throws  ClassCastException
      chkSet.add(10); //throws ClassCastException
   }
```

```
}
```
Output:
```
Checked set content: [two, one, three, four]
Exception in thread "main" java.lang.ClassCastException: Attempt to
insert class java.lang.Integer element into
collection with element type class java.lang.String
        at
java.util.Collections$CheckedCollection.typeCheck(Collections.java:2202)
        at
java.util.Collections$CheckedCollection.add(Collections.java:2243)
        at
com.java2novice.collections.MyCheckedSet.main(MyCheckedSet.java:22)
```

352. Write an example for Collections.checkedMap() method.

```
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

public class MyCheckedMap {

   public static void main(String a[]){

      Map myMap = new HashMap();
      myMap.put("one", 1);
      myMap.put("two", 2);
      myMap.put("three", 3);
      myMap.put("four", 4);
      Map chkMap = Collections.checkedMap(myMap, String.class,
Integer.class);
      System.out.println("Checked map content: "+chkMap);
      //you can add any type of elements to myMap object
      myMap.put(10, "ten");
      //you cannot add any type of elements to chkMap object, doing so
      //throws ClassCastException
```

```
        chkMap.put(10, "ten"); //throws ClassCastException
    }
}
```

Output:

```
Checked map content: {two=2, one=1, three=3, four=4}
Exception in thread "main" java.lang.ClassCastException: Attempt to
insert class java.lang.Integer key into
                                        collection with key type class
java.lang.String
        at
java.util.Collections$CheckedMap.typeCheck(Collections.java:2547)
        at java.util.Collections$CheckedMap.put(Collections.java:2579)
        at
com.java2novice.collections.MyCheckedMap.main(MyCheckedMap.java:2
2)
```

353. How to check there in no common element between two list objects by using Collections.disjoint() method?

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class MyListDisjoint {

    public static void main(String a[]){

        List<String> sl = new ArrayList<String>();
        sl.add("apple");
        sl.add("java");
        sl.add("c++");
        sl.add("unix");
        sl.add("orange");
        sl.add("airtel");
```

```
        List<String> tl = new ArrayList<String>();
        tl.add("job");
        tl.add("oracle");
        tl.add("jungle");
        tl.add("cricket");
        boolean isCommon = Collections.disjoint(sl,tl);
        System.out.println("Does not found any common elements?
"+isCommon);
        tl.add("java");
        isCommon = Collections.disjoint(sl,tl);
        System.out.println("Does not found any common elements?
"+isCommon);
    }
}
```

Output:
Does not found any common elements? true
Does not found any common elements? False

354. How to rotate elements in the list by specified distance?

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class MyListRotate {

    public static void main(String a[]){

        List<String> list = new ArrayList<String>();
        list.add("java");
        list.add("c");
        list.add("c++");
        list.add("unix");
        list.add("perl");
```

```
        list.add("php");
        list.add("javascript");
        list.add("ruby");
        list.add(".net");
        System.out.println(list);
        Collections.rotate(list, 3);
        System.out.println("List after rotation:");
        System.out.println(list);
    }
}
```

Output:

```
[java, c, c++, unix, perl, php, javascript, ruby, .net]
List after rotation:
[javascript, ruby, .net, java, c, c++, unix, perl, php]
```

NETWORKING

355. How do you represent a URL in Java programming language?

The Java API provides the URL class which can be used to represent the URL address. You can create the URL object if you have the URL address string. The URL class provides getter methods to get the components of the URL such as host name, port, path, query parameters etc.

Example:

```
String urlString = 'http://www.kodnest.com';
 URL url = new URL(urlString);
```

356. How do you connect to a URL resource in Java programming language?

The Java API provides the 'URLConnecton' class which can be used to create a connection to a URL. If you have a URL object, you can get the URLConnection object by calling openConnection() method on the URL object. Once you have the URLConnection object you can connect to the URL resource by calling the connect() method on the URLConnection object. You can use the URLRequest object to setup parameters and properties that you may need for making the URL connection.

Example:

```
String urlString = 'http://www.kodnest.com';
URL myUrl = new URL(urlString);
URLConnection myUrlConnection = myUrl.openConnection();
myUrlConnection.connect();
```

357. What are the key steps in reading from a URL connection?

1. Create the URL object
2. Create URLConnection object
3. Open connection to URL
4. Get input stream from connection
5. Read from input stream
6. Close input stream

358. What are the key steps in writing to a URL connection?

1. Create the URL object
2. Create URLConnection object
3. Open connection to URL
4. Get output stream from connection
5. Write to output stream
6. Close output stream

359. What are sockets? How are sockets represented in the Java programming language?

Sockets are end points in the communication link between a client program and a server program exchanging data over a network.

On the server side, a socket is bound to a specific port number. The server listens to the socket, waiting for a client to make a connection request.If a connection from a client is successful, the existing socked is used to communicate with that client. In addition a new socket is created and ties to the same port so that the server can listen to new connections from other clients.A new

On the client side: The client makes a connection request to the server, specific to the port number that the server socket is tied to. To successfully connect to the server, the client has to identify itself to the server along with its port number. Hence the client binds itself to a local port before making the connection request.If the connection is successful, a socket is created on the client side that is tied to the port and is used for communicating with the server.

The Java programming language provides two classes to represent sockets. Class 'java.net.Socket' represents a socket at the client side. Class 'java.net.ServerSocket' represents a socket on the server side.

360. What are the key steps in reading writing to sockets?

1. Open a socket
2. Open an input stream and output stream to a socket
3. Read from and write to the stream
4. Close the streams
5. Close the socke

361. What is the difference between TCP and UDP protocols?

TCP is a protocol that provides a reliable, point-to-point communication channel that client-server application use to communicate with each other. To communicate over TCP, a client program and server program must first establish a connection to each other through sockets at each end of the communication channel. To communicate, the client and server reads from and writes to the sockets bound to the connection.

Like TCP, UDP is protocol that provides a communication channel that client-server applications use to communicate with each other. But unlike TCP, the message content and arrival time of communication sent via UDP are not guaranteed. In UDP messages are sent via datagrams, which are independent, self-contained packets of data.

362. What is datagram? What are some key classes defined in the Java programming language to work with datagrams?

Datagram is an independent, self-contained packet of information send over the network between server and client programs in UDP protocol. The delivery of datagrams to their destinations in not guaranteed. The order of arrival of datagrams to their destinations is not guaranteed. Datagrams can be send or broadcast to multiple recipients.

Java programming language provides three main classes that can be used to program datagrams - java.net.DatagramPacket, java.net.DatagramSocket, java.net.MultigramSocket

363. What is a network interface?

A network interface is the point of interconnection between a computer and a private or public network. A network interface is generally a network interface card (NIC). Network interfaces can either have a physical form or can be implemented in software.

The Java networking API provides the java.net.NetworkInterface class which represents both these types of interfaces.

364. How do you get a list of IP addresses that are assigned to a network interface?

You can get a list of IP addresses that are assigned to a network interface using the NetworkInterface class. You can obtain this information from a NetworkInterface instance by using one of two methods.

1. getInetAddresses() - returns an Enumeration of InetAddress.
2. 2. getInterfaceAddresses() - returns a list of java.net.InterfaceAddress instances. This method is used when you need more information about an interface address beyond its IP address such as it subnet mask.

365. Write an Example of Java Socket Programming

MyServer.java

```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
}catch(Exception e){System.out.println(e);}
```

```
}
}
```

MyClient.java

```
import java.io.*;
import java.net.*;
public class MyClient
 {
public static void main(String[] args)
{
try{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new
DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```

366. Write an Example of Java Socket Programming which has Read-Write both side.

MyServer.java

```
import java.net.*;
import java.io.*;
class MyServer
{
public static void main(String args[])throws Exception
{
ServerSocket ss=new ServerSocket(3333);
Socket s=ss.accept();
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new
DataOutputStream(s.getOutputStream());
```

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

String str="",str2="";
while(!str.equals("stop")){
str=din.readUTF();
System.out.println("client says: "+str);
str2=br.readLine();
dout.writeUTF(str2);
dout.flush();
}
din.close();
s.close();
ss.close();
}
}
```

MyClient.java

```
import java.net.*;
import java.io.*;
class MyClient
{
public static void main(String args[])throws Exception
{
Socket s=new Socket("localhost",3333);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new
DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));

String str="",str2="";
while(!str.equals("stop")){
str=br.readLine();
dout.writeUTF(str);
dout.flush();
str2=din.readUTF();
System.out.println("Server says: "+str2);
```

```
}
dout.close();
s.close();
}
}
```

SOME COMMONLY ASKED JAVA PROGRAMS:

1.Java Program to print Fibonacci Series

```
import java.util.Scanner;
public class FibonacciCalculator {
   public static void main(String args[]) {
      //input to print Fibonacci series upto how many numbers
       System.out.println("Enter number upto which Fibonacci series to
print: ");
       int number = new Scanner(System.in).nextInt();
       System.out.println("Fibonacci series upto " + number +" numbers : ");
       //printing Fibonacci series upto number
       for(int i=1; i<=number; i++){
          System.out.print(fibonacci2(i) +" ");
       }
     }
     /*
    * Java program for Fibonacci number using recursion.
    * This program uses tail recursion to calculate Fibonacci number for a
given number
    * @return Fibonacci number
    */
   public static int fibonacci(int number){
       if(number == 1 || number == 2){
          return 1;
       }
        return fibonacci(number-1) + fibonacci(number -2); //tail recursion
   }
     /*
    * Java program to calculate Fibonacci number using loop or Iteration.
    * @return Fibonacci number
    */
   public static int fibonacci2(int number){
       if(number == 1 || number == 2){
          return 1;
       }
       int fib01=1, fib02=1, fibonacci=1;
       for(int i= 3; i<= number; i++){
```

```
            //Fibonacci number is sum of previous two Fibonacci number
            fibonacci = fibo1 + fibo2;
            fibo1 = fibo2;
            fibo2 = fibonacci;
          }
        return fibonacci; //Fibonacci number
    }
 }
```

Output:

```
Enter number upto which Fibonacci series to print:
12
Fibonacci series upto 12 numbers :
1 1 2 3 5 8 13 21 34 55 89 144
```

2. Find factorial of a number in Java using recursion and iteration :

```
/**
 * Simple Java program to find the factorial of a number using recursion
and iteration.
 * Iteration will use for loop while recursion will call method itself
 */
public class FactorialInJava{
   public static void main(String args[]) {
     //finding factorial of a number in Java using recursion - Example
     System.out.println("factorial of 5 using recursion in Java is: " +
factorial(5));
     //finding factorial of a number in Java using Iteration - Example
      System.out.println("factorial of 6 using iteration in Java is: " + fact(6));
   }
    /*
    * Java program example to find factorial of a number using recursion
    * @return factorial of number
    */
   public static int factorial(int number){
       //base case
       if(number == 0){
           return 1;
       }
```

```
        return number*factorial(number -1); //is this tail-recursion?
    }
     /*
     * Java program example to calculate factorial using while loop or
iteration
     * @return factorial of number
     */
     public static int fact(int number){
       int result = 1;
       while(number != 0){
          result = result*number;
          number--;
       }
         return result;
    }
}
```

Output:

factorial of 5 using recursion in Java is: 120

factorial of 6 using iteration in Java is: 720

3. Program to reverse a String

```
public class StringReverseExample {
      public static void main(String args[]) {
        //quick wasy to reverse String in Java - Use StringBuffer
        String word = "HelloWorld";
        String reverse = new StringBuffer(word).reverse().toString();
        System.out.printf(" original String : %s ,
              reversed String %s  %n", word, reverse);
        //another quick to reverse String in Java - use StringBuilder
        word = "WakeUp";
        reverse = new StringBuilder(word).reverse().toString();
        System.out.printf(" original String : %s ,
            reversed String %s %n", word, reverse);
        // one way to reverse String without using
        // StringBuffer or StringBuilder is writing
        // own utility method
        word = "Band";
        reverse = reverse(word);
        System.out.printf(" original String : %s ,
```

```
                    reversed String %s %n", word, reverse);
    }
    public static String reverse(String source){
        if(source == null || source.isEmpty()){
            return source;
        }
        String reverse = "";
        for(int i = source.length() -1; i>=0; i--){
            reverse = reverse + source.charAt(i);
        }
        return reverse;
    }
}
```

Output:

```
original String: HelloWorld, reversed String dlroWolleH
original String: WakeUp, reversed String pUekaW
original String: Band, reversed String dnaB
```

4. Removing duplicates from integer array:

```
import java.util.Arrays;
public class RemoveDuplicates {
    public static void main(String args[]) {
        int[][] test = new int[][]{
            {1, 1, 2, 2, 3, 4, 5},
            {1, 1, 1, 1, 1, 1, 1},
            {1, 2, 3, 4, 5, 6, 7},
            {1, 2, 1, 1, 1, 1, 1},};
        for (int[] input : test) {
            System.out.println("Array with Duplicates      : " +
Arrays.toString(input));
            System.out.println("After removing duplicates   : " +
Arrays.toString(removeDuplicates(input)));
        }
    }
    /*
     * Method to remove duplicates from array in Java, without using
     * Collection classes e.g. Set or ArrayList. Algorithm for this
     * method is simple, it first sort the array and then compare adjacent
     * objects, leaving out duplicates, which is already in the result.
```

```
    */
    public static int[] removeDuplicates(int[] numbersWithDuplicates) {
        // Sorting array to bring duplicates together
        Arrays.sort(numbersWithDuplicates);
        int[] result = new int[numbersWithDuplicates.length];
        int previous = numbersWithDuplicates[0];
        result[0] = previous;
        for (int i = 1; i < numbersWithDuplicates.length; i++) {
            int ch = numbersWithDuplicates[i];
            if (previous != ch) {
                result[i] = ch;
            }
            previous = ch;
        }
        return result;
    }
}
```

Output :

```
Array with Duplicates      : [1, 1, 2, 2, 3, 4, 5]
After removing duplicates   : [1, 0, 2, 0, 3, 4, 5]
Array with Duplicates      : [1, 1, 1, 1, 1, 1, 1]
After removing duplicates   : [1, 0, 0, 0, 0, 0, 0]
Array with Duplicates      : [1, 2, 3, 4, 5, 6, 7]
After removing duplicates   : [1, 2, 3, 4, 5, 6, 7]
Array with Duplicates      : [1, 2, 1, 1, 1, 1, 1]
After removing duplicates   : [1, 0, 0, 0, 0, 0, 2]
```

5. Java Program to find the largest and smallest element in array:

```
import java.util.Arrays;
public class MaximumMinimumArrayDemo{
    public static void main(String args[]) {
        largestAndSmallest(new int[]{-20, 34, 21, -87, 92,
                        Integer.MAX_VALUE});
        largestAndSmallest(new int[]{10, Integer.MIN_VALUE, -2});
        largestAndSmallest(new int[]{Integer.MAX_VALUE, 40,
                        Integer.MAX_VALUE});
        largestAndSmallest(new int[]{1, -1, 0});
    }
```

```
    public static void largestAndSmallest(int[] numbers) {
        int largest = Integer.MIN_VALUE;
        int smallest = Integer.MAX_VALUE;
        for (int number : numbers) {
            if (number > largest) {
                largest = number;
            } else if (number < smallest) {
                smallest = number;
            }
        }

        System.out.println("Given integer array : " +
Arrays.toString(numbers));
        System.out.println("Largest number in array is : " + largest);
        System.out.println("Smallest number in array is : " + smallest);
    }
}
```

Output:

```
Given integer array : [-20, 34, 21, -87, 92, 2147483647]
Largest number in array is : 2147483647
Smallest number in array is : -87
Given integer array : [10, -2147483648, -2]
Largest number in array is : 10
Smallest number in array is : -2147483648
Given integer array : [2147483647, 40, 2147483647]
Largest number in array is : 2147483647
Smallest number in array is : 40
Given integer array : [1, -1, 0]
Largest number in array is : 1
Smallest number in array is : -1
```

6. How to reverse number in Java

```
import java.util.Scanner;
public class ReverseNumberExample {
    public static void main(String args[]) {
       //input number to reverse
        System.out.println("Please enter number to be reversed using Java
program: ");
        int number = new Scanner(System.in).nextInt();
```

```
        int reverse = reverse(number);
        System.out.println("Reverse of number: " + number + " is " +
reverse(number));
    }
    public static int reverse(int number){
        int reverse = 0;
        int remainder = 0;
        do{
            remainder = number%10;
            reverse = reverse*10 + remainder;
            number = number/10;
        }while(number > 0);
                return reverse;
    }
}
```

Output:

```
Please enter number to be reversed using Java program:
1234
Reverse of number: 1234 is 4321
```

7. Java Program to implement QuickSort Algorithm

```
import java.util.Arrays;
public class QuickSortDemo{
    public static void main(String args[]) {
        // unsorted integer array
        int[] unsorted = {6, 5, 3, 1, 8, 7, 2, 4};
        System.out.println("Unsorted array :" + Arrays.toString(unsorted));
        QuickSort algorithm = new QuickSort();
        // sorting integer array using quicksort algorithm
        algorithm.sort(unsorted);
        // printing sorted array
        System.out.println("Sorted array :" + Arrays.toString(unsorted));
    }
}
class QuickSort {
    private int input[];
    private int length;
    public void sort(int[] numbers) {
        if (numbers == null || numbers.length == 0) {
```

```
            return;
        }
        this.input = numbers;
        length = numbers.length;
        quickSort(0, length - 1);
    }
    /*
     * This method implements in-place quicksort algorithm recursively.
     */
    private void quickSort(int low, int high) {
        int i = low;
        int j = high;
        // pivot is middle index
        int pivot = input[low + (high - low) / 2];
        // Divide into two arrays
        while (i <= j) {
            while (input[i] < pivot) {
                i++;
            }
            while (input[j] > pivot) {
                j--;
            }
            if (i <= j) {
                swap(i, j);
                // move index to next position on both sides
                i++;
                j--;
            }
        }
        // calls quickSort() method recursively
        if (low < j) {
            quickSort(low, j);
        }
        if (i < high) {
            quickSort(i, high);
        }
    }
    private void swap(int i, int j) {
        int temp = input[i];
```

```
        input[i] = input[j];
        input[j] = temp;
    }
}
```

Output :

```
Unsorted array :[6, 5, 3, 1, 8, 7, 2, 4]
Sorted array :[1, 2, 3, 4, 5, 6, 7, 8]
```

8. Insertion Sort implementation in Java

```
import java.util.Arrays;
public class Pattern {
  public static void main(String args[]) {
    // unsorted integer array
    int[] unsorted = { 32, 23, 45, 87, 92, 31, 19 };
    System.out.println("integer array before sorting : "
                       + Arrays.toString(unsorted));
    insertionSort(unsorted);
    System.out.println("integer array after sorting : "
                       + Arrays.toString(unsorted));
  }
    public static void insertionSort(int[] unsorted) {
    for (int i = 1; i < unsorted.length; i++) {
      int current = unsorted[i];
      int j = i;
      // create right place by moving elements
      while (j > 0 && unsorted[j - 1] > current) {
        // move
        unsorted[j] = unsorted[j - 1];
        j--;
      }
      // found the right place, insert now
      unsorted[j] = current;
    }
  }
}
```

Output

```
integer array before sorting : [32, 23, 45, 87, 92, 31, 19]
integer array after sorting : [19, 23, 31, 32, 45, 87, 92]
```

9. Bubble Sort Implementation in Java

```
import java.util.Arrays;
public class BubbleSort{
  public static void main(String args[]) {
      bubbleSort(new int[] { 20, 12, 45, 19, 91, 55 });
      bubbleSort(new int[] { -1, 0, 1 });
      bubbleSort(new int[] { -3, -9, -2, -1 });
  }
  /*
   * This method sort the integer array using bubble sort algorithm
   */
  public static void bubbleSort(int[] numbers) {
      System.out.printf("Unsorted array in Java :%s %n",
Arrays.toString(numbers));
      for (int i = 0; i < numbers.length; i++) {
         for (int j = numbers.length -1; j > i; j--) {
            if (numbers[j] < numbers[j - 1]) {
               swap(numbers, j, j-1);
            }
         }
      }
      System.out.printf("Sorted Array using Bubble sort algorithm :%s %n",
            Arrays.toString(numbers));
  }
  /*
   * Utility method to swap two numbers in array
   */
  public static void swap(int[] array, int from, int to){
      int temp = array[from];
      array[from] = array[to];
      array[to] = temp;
  }
}
```

Output

```
Unsorted array in Java : [20, 12, 45, 19, 91, 55]
Sorted Array using Bubble sort algorithm : [12, 19, 20, 45, 55, 91]
Unsorted array in Java : [-1, 0, 1]
Sorted Array using Bubble sort algorithm : [-1, 0, 1]
Unsorted array in Java : [-3, -9, -2, -1]
```

Sorted Array using Bubble sort algorithm : [-9, -3, -2, -1]

10. Java Program to Print All Permutation of a String

```
public class StringPermutations {
   public static void main(String args[]) {
      permutation("123");
   }
  public static void permutation(String input){
        permutation("", input);
  }
  private static void permutation(String perm, String word) {
      if (word.isEmpty()) {
         System.err.println(perm + word);

      } else {
         for (int i = 0; i &lt; word.length(); i++) {
            permutation(perm + word.charAt(i), word.substring(0, i)
                              + word.substring(i + 1, word.length()));
         }
      }

   }
}
```

Output:

```
123
132
213
231
312
321
```

11. Java program to count number of occurrence of any character on String:

```
import org.springframework.util.StringUtils;
public class CountCharacters {
```

```
   public static void main(String args[]) {

      String input = "Today is Monday"; //count number of "a" on this
String.

      int count = StringUtils.countOccurrencesOf(input, "a");

      System.out.println("count of occurrence of character 'a' on String: " +
            " Today is Monday' using Spring StringUtils " + count);

      int number =
org.apache.commons.lang.StringUtils.countMatches(input, "a");
      System.out.println("count of character 'a' on String: 'Today is
Monday' using commons StringUtils " + number);

      //counting occurrence of character with loop
      int charCount = 0;
      for(int i =0 ; i<input.length(); i++){
         if(input.charAt(i) == 'a'){
            charCount++;
         }
      }
      System.out.println("count of character 'a' on String: 'Today is
Monday' using for loop  " + charCount);
      charCount = 0; //resetting character count
      for(char ch: input.toCharArray()){
         if(ch == 'a'){
            charCount++;
         }
      }
      System.out.println("count of character 'a' on String: 'Today is
Monday' using for each loop  " + charCount);
   }
}
```

Output
count of occurrence of character 'a' on String: 'Today is Monday' using Spring StringUtils 2

count of character 'a' on String: 'Today is Monday' using commons StringUtils 2
count of character 'a' on String: 'Today is Monday' using for loop 2
count of character 'a' on String: 'Today is Monday' using for each loop 2

12. Java Program to count vowels and consonants in String

```
import java.util.Scanner;
public class VowelCounter {

    public static void main(String args[]) {
        System.out.println("Please enter some text");
        Scanner reader = new Scanner(System.in);

     String input = reader.nextLine();
     char[] letters = input.toCharArray();
     int count = 0;

     for (char c : letters) {
         switch (c) {
            case 'a':
               case 'e':
               case 'i':
               case 'o':
               case 'u':
                  count++;
        break;
        default:
               // no count increment
        }

      }
          System.out.println("Number of vowels in String [" + input + "] is : "
+ count);
    }

}
```

Output:

```
Please enter some text
How many vowels in this String
Number of vowels in String [How many vowels in this String] is : 7
```

13. Count number of occurrence of any character on String:

```
import org.springframework.util.StringUtils;
public class CountCharacters {
   public static void main(String args[]) {
       String input = "Today is Monday"; //count number of "a" on this
String.
       //Using Spring framework StringUtils class for finding occurrence of
another String
      int count = StringUtils.countOccurrencesOf(input, "a");
     System.out.println("count of occurrence of character 'a' on String: " +
            " Today is Monday' using Spring StringUtils " + count);
      //Using Apache commons lang StringUtils class
      int number =
org.apache.commons.lang.StringUtils.countMatches(input, "a");
      System.out.println("count of character 'a' on String: 'Today is
Monday' using commons StringUtils " + number);

      //counting occurrence of character with loop
      int charCount = 0;
      for(int i =0 ; i<input.length(); i++){
         if(input.charAt(i) == 'a'){
            charCount++;
         }
      }
      System.out.println("count of character 'a' on String: 'Today is
Monday' using for loop  " + charCount);

      //a more elegant way of counting occurrence of character in String
using foreach loop

      charCount = 0; //resetting character count
      for(char ch: input.toCharArray()){
         if(ch == 'a'){
            charCount++;
```

```
            }
        }
        System.out.println("count of character 'a' on String: 'Today is
Monday' using for each loop  " + charCount);
    }
}
```

Output
count of occurrence of character 'a' on String: 'Today is Monday' using Spring StringUtils 2
count of character 'a' on String: 'Today is Monday' using commons StringUtils 2
count of character 'a' on String: 'Today is Monday' using for loop 2
count of character 'a' on String: 'Today is Monday' using for each loop 2

14. Removing a given Character From String Recursively

```
import java.util.ArrayList;
import java.util.List;

public class RemoveCharFromString {

    public static String remove(String word, char unwanted){
        StringBuilder sb = new StringBuilder();
        char[] letters = word.toCharArray();

        for(char c : letters){
            if(c != unwanted ){
                sb.append(c);
            }
        }

        return sb.toString();
    }

    public static String removeRecursive(String word, char ch){
        int index = word.indexOf(ch);
        if(index == -1){
            return word;
```

```
        }
        return removeRecursive(word.substring(0, index) +
word.substring(index +1, word.length()), ch);
    }
}
```

15. Java Program to find Repeated Characters of String

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import java.util.Set;


public   class   FindDuplicateCharacters{

    public static void main(String args[]) {
        printDuplicateCharacters("Programming");
        printDuplicateCharacters("Combination");
        printDuplicateCharacters("Java");
    }

    /*
     * Find all duplicate characters in a String and print each of them.
     */
    public static void printDuplicateCharacters(String word) {
        char[] characters = word.toCharArray();

        // build HashMap with character and number of times they appear in
String
        Map<Character, Integer> charMap = new HashMap<Character,
Integer>();
        for (Character ch : characters) {
            if (charMap.containsKey(ch)) {
                charMap.put(ch, charMap.get(ch) + 1);
            } else {
                charMap.put(ch, 1);
            }
        }
```

```
        // Iterate through HashMap to print all duplicate characters of String
        Set<Map.Entry<Character, Integer>> entrySet = charMap.entrySet();
        System.out.printf("List of duplicate characters in String '%s' %n",
word);
        for (Map.Entry<Character, Integer> entry : entrySet) {
            if (entry.getValue() > 1) {
                System.out.printf("%s : %d %n", entry.getKey(),
entry.getValue());
            }
        }
    }

}
```

Output

```
List of duplicate characters in String 'Programming'
g : 2
r : 2
m : 2
List of duplicate characters in String 'Combination'
n : 2
o : 2
i : 2
List of duplicate characters in String 'Java'
```

16. Java Program to calculate GCD of two numbers

```
public class GCDExample {

    public static void main(String args[]){

        //Enter two number whose GCD needs to be calculated.
        Scanner scanner = new Scanner(System.in);
        System.out.println("Please enter first number to find GCD");
        int number1 = scanner.nextInt();
        System.out.println("Please enter second number to find GCD");
        int number2 = scanner.nextInt();
```

```
        System.out.println("GCD of two numbers " + number1 +" and "
                        + number2 +" is :" + findGCD(number1,number2));


    }

    /*
     * Java method to find GCD of two number using Euclid's method
     * @return GDC of two numbers in Java
     */
    private static int findGCD(int number1, int number2) {
        //base case
        if(number2 == 0){
            return number1;
        }
        return findGCD(number2, number1%number2);
    }

}
```

Output:
Please enter first number to find GCD
54
Please enter second number to find GCD
24
GCD of two numbers 54 and 24 is :6

17. Check if a year is a leap year in Java

```
import java.util.Calendar;

public class LeapYearProgram {

    public static void main(String args[]) {

        //Testing some leap and non leap year using Java library code
        System.err.println("Is 2000 a leap year ? : " + isLeapYear(2000));
        System.err.println("Is 2012 a leap year ? : " + isLeapYear(2012));
        System.err.println("Is 1901 a leap year ? : " + isLeapYear(1901));
```

```
        System.err.println("Is 1900 a leap year ? : " + isLeapYear(1900));


        //Checking leap year without using library or API and applying logic
        System.err.println("Does 2000 a leap year : " + doesLeapYear(2000));
        System.err.println("Does 2012 a leap year : " + doesLeapYear(2012));
        System.err.println("Does 1901 a leap year : " + doesLeapYear(1901));
        System.err.println("Does 1900 a leap year : " + doesLeapYear(1900));
    }
     /*
     * This method checks whether a year is leap or not by using Java Date
     * and Time API. Calendar class has utility method to return maximum
     * number of days in a year which can be used to check if its
     * greater than 365 or not
     */
    public static boolean isLeapYear(int year){
        Calendar cal = Calendar.getInstance(); //gets Calendar based on local
timezone and locale
        cal.set(Calendar.YEAR, year); //setting the calendar year
        int noOfDays = cal.getActualMaximum(Calendar.DAY_OF_YEAR);

        if(noOfDays > 365){
            return true;
        }

        return false;
    }

    /*
     * This method uses standard logic to check leap year in Java.
     * A year is a leap year if its multiple of 400 or multiple of 4 but not 100
     */
    public static boolean doesLeapYear(int year){
        return (year%400 == 0) || ((year%100) != 0 && (year%4 == 0));
    }
}
```

Output:
Is 2000 a leap year ? : true

```
Is 2012 a leap year ? : true
Is 1901 a leap year ? : false
Is 1900 a leap year ? : false
Does 2000 a leap year : true
Does 2012 a leap year : true
Does 1901 a leap year : false
Does 1900 a leap year : false
```

18. Java program to check if String is anagram

```
import java.util.Arrays;
public class AnagramCheck {

   /*
    * One way to find if two Strings are anagram in Java. This method
    * assumes both arguments are not null and in lowercase.
    *
    * @return true, if both String are anagram
    */
   public static boolean isAnagram(String word, String anagram){
      if(word.length() != anagram.length()){
         return false;
      }

      char[] chars = word.toCharArray();

      for(char c : chars){
         int index = anagram.indexOf(c);
         if(index != -1){
            anagram = anagram.substring(0,index) +
anagram.substring(index +1, anagram.length());
         }else{
            return false;
         }
      }

      return anagram.isEmpty();
   }
```

```
    /*
     * Another way to check if two Strings are anagram or not in Java
     * This method assumes that both word and anagram are not null and
lowercase
     * @return true, if both Strings are anagram.
     */
    public static boolean iAnagram(String word, String anagram){
        char[] charFromWord = word.toCharArray();
        char[] charFromAnagram = anagram.toCharArray();
        Arrays.sort(charFromWord);
        Arrays.sort(charFromAnagram);
        return Arrays.equals(charFromWord, charFromAnagram);
    }
    public static boolean checkAnagram(String first, String second){
        char[] characters = first.toCharArray();
        StringBuilder sbSecond = new StringBuilder(second);

        for(char ch : characters){
            int index = sbSecond.indexOf("" + ch);
            if(index != -1){
                sbSecond.deleteCharAt(index);
            }else{
                return false;
            }
        }

        return sbSecond.length()==0 ? true : false;
    }
}
```

19. Program to print all leaf nodes of this binary tree in Java:

```
public class Main {

  public static void main(String[] args) throws Exception {

    // let's create a binary tree
    TreeNode d = new TreeNode("d");
    TreeNode e = new TreeNode("e");
```

```
  TreeNode g = new TreeNode("g");
  TreeNode k = new TreeNode("k");

  TreeNode c = new TreeNode("c", d, null);
  TreeNode h = new TreeNode("h", k, null);

  TreeNode b = new TreeNode("b", c, e);
  TreeNode f = new TreeNode("f", g, h);

  TreeNode root = new TreeNode("a", b, f);

  // print all leaf nodes of binary tree using recursion
  System.out
     .println("Printing all leaf nodes of binary tree in Java (recursively)");
  printLeaves(root);

}

/**
 * A class to represent a node in binary tree
 */
private static class TreeNode {
  String value;
  TreeNode left;
  TreeNode right;

  TreeNode(String value) {
    this.value = value;
  }

  TreeNode(String data, TreeNode left, TreeNode right) {
    this.value = data;
    this.left = left;
    this.right = right;
  }
  boolean isLeaf() {
    return left == null ? right == null : false;
  }
}
```

```
  /**
   * Java method to print leaf nodes using recursion
   *
   * @param root
   *
   */
  public static void printLeaves(TreeNode node) {
    // base case
    if (node == null) {
      return;
    }
    if (node.isLeaf()) {
      System.out.printf("%s ", node.value);
    }
    printLeaves(node.left);
    printLeaves(node.right);

  }
}
```

Output
Printing all leaf nodes of binary tree in Java (recursively)
d e g k

20. Program for transposing a Matrix in Java

```
import java.util.Scanner;

/*
 * Java Program to transpose a Matrix. When you transpose
 * a matrix rows are replaced by columns. For example,
 * The transpose of a matrix is a new matrix whose rows are the columns
of the original.
 */
public class MatrixTransposeDemo {

  public static void main(String[] args) {
```

```
    System.out.println("Welcome to Java program to transpose a Matrix");
    Scanner scnr = new Scanner(System.in);

    System.out.println("Please enter details of matrix");
    System.out.print("Please Enter number of rows: ");
    int row1 = scnr.nextInt();
    System.out.print("Please Enter number of columns: ");
    int column1 = scnr.nextInt();
    System.out.println();
    System.out.println("Enter first matrix elements");
    Matrix first = new Matrix(row1, column1);
    first.read(scnr);

    System.out.println("original matrix: ");
    first.print();

    // let's transpose the matrix now
    first.transpose();

    System.out.println("transpose of the matrix is ");
    first.print();
    scnr.close();

  }

}

/*
 * Java class to represent a Matrix. It uses a two dimensional array to
 * represent a Matrix.
 */
class Matrix {
  private int rows;
  private int columns;
  private int[][] data;

  public Matrix(int row, int column) {
    this.rows = row;
```

```
  this.columns = column;
  data = new int[rows][columns];
 }

 public Matrix(int[][] data) {
  this.data = data;
  this.rows = data.length;
  this.columns = data[0].length;
 }

 public int getRows() {
  return rows;
 }

 public int getColumns() {
  return columns;
 }

 /**
  * fills matrix from data entered by user in console
  *
  * @param rows
  * @param columns
  */
 public void read(Scanner s) {
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < columns; j++) {
      data[i][j] = s.nextInt();
    }
  }

 }

 /**
  * This method will transpose this matrix
  *
  * @return
  */
 public void transpose() {
```

```
    int[][] temp = new int[columns][rows];
    for (int i = 0; i < rows; i++) {
      for (int j = 0; j < columns; j++) {
        temp[j][i] = data[i][j];
      }
    }
    data = temp;
  }

  /**
   *
   * @param matrix
   */
  public void print() {
    for (int i = 0; i < rows; i++) {
      for (int j = 0; j < columns; j++) {
        System.out.print(data[i][j] + " ");
      }
      System.out.println();
    }
  }

}
```

Output

```
Welcome to Java program to transpose a Matrix
Please enter details of matrix
Please Enter number of rows: 2
Please Enter number of columns: 2

Enter first matrix elements
1
2
3
4
original matrix:
1 2
```

```
3 4
transpose of the matrix is
1 3
2 4
```

21. Java Program to add and subtract two matrices in Java

```
import java.util.Scanner;

/*
 * Java Program to add and subtract two matrices.
 * A matrix can be represented as two dimensional array in Java
 */
public class MatrixAdditionSubtractionDemo {

  public static void main(String[] args) {

    System.out
        .println("Welcome to Java program for calculating sum and
difference of two matrices");

    // we need a Scanner to read input from Console
    Scanner scnr = new Scanner(System.in);

    System.out.print("Please Enter number of rows: ");
    int rows = scnr.nextInt();

    System.out.print("Please Enter number of columns: ");
    int columns = scnr.nextInt();
    System.out.println();

    System.out.println("Please Enter first matrix");
    int[][] a = read(scnr, rows, columns);
    System.out.println();

    System.out.println("Please Enter second matrix");
    int[][] b = read(scnr, rows, columns);

    scnr.close();
```

```
  // adding two matrices
  int[][] sum = add(a, b);

  // subtracting two  matrices
  int[][] difference1 = subtract(a, b);
  int[][] difference2 = subtract(b, a);

  System.out.println("The sum of two matrices is: ");
  System.out.println("A + B =");
  printMatrix(sum);

  System.out.println("The differnece of two matrices is: ");
  System.out.println("A - B =");
  printMatrix(difference1);

  System.out.println("Subtraction of matrix in opposite order");
  System.out.println("B - A =");
  printMatrix(difference2);

  scnr.close();
}

/**
 * a method to populate a matrix by reading input from console in Java
 *
 * @param rows
 * @param columns
 * @return matrix filled by user input from console
 */
public static int[][] read(Scanner s, int rows, int columns) {
  int[][] result = new int[rows][columns];;
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < columns; j++) {
      System.out.println("Enter value of [" + (i+1) + "][" + (j+1) +"]");
      result[i][j] = s.nextInt();
    }
  }
  return result;
```

```
}

/**
 * Java Program to calculate sum of two matrices
 *
 * @param a
 * @param b
 * @return return sum of given matrices
 */
public static int[][] add(int[][] a, int[][] b) {
  int rows = a.length;
  int columns = a[0].length;
  int[][] result = new int[rows][columns];
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < columns; j++) {
      result[i][j] = a[i][j] + b[i][j];
    }
  }
  return result;
}

/**
 * Java Program to calculate difference of two matrices
 *
 * @param a
 * @param b
 * @return difference of given matrix
 */
public static int[][] subtract(int[][] a, int[][] b) {
  int rows = a.length;
  int columns = a[0].length;
  int[][] result = new int[rows][columns];
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < columns; j++) {
      result[i][j] = a[i][j] - b[i][j];
    }
  }
  return result;
}
```

```
  /**
   * a Java method to print result in matrix format.
   *
   * @param matrix
   */
  public static void printMatrix(int[][] matrix) {
    int rows = matrix.length;
    int columns = matrix[0].length;
    for (int i = 0; i < rows; i++) {
      for (int j = 0; j < columns; j++) {
        System.out.print(matrix[i][j] + " ");
      }
      System.out.println();
    }
  }
}
```

Output

```
Welcome to Java program for calculating sum and difference of two matrices
Please Enter number of rows: 2
Please Enter number of columns: 2

Please Enter first matrix
Enter value of [1][1]
1
Enter value of [1][2]
2
Enter value of [2][1]
3
Enter value of [2][2]
4

Please Enter second matrix
Enter value of [1][1]
5
Enter value of [1][2]
6
```

```
Enter value of [2][1]
7
Enter value of [2][2]
8
The sum of two matrices is:
A + B =
6 8
10 12
The difference of two matrices is:
A - B =
-4 -4
-4 -4
Subtraction of matrix in opposite order
B - A =
4 4
4 4
```

22. Java program to calculate product of Two matrices

```
import java.util.Scanner;
public  class MatrixMultiplication{

   public static void main(String args[]) {

      Scanner cmd = new Scanner(System.in);
      System.out.println("Enter the number of rows and columns of
                  first matrix");

      int rowsOfFirstMatrix = cmd.nextInt();
      int columnsOfFirstMatrix = cmd.nextInt();
      int[][] aMatrix = new int[rowsOfFirstMatrix][columnsOfFirstMatrix];

      System.out.println("Enter the elements of first matrix");
      for (int i = 0; i &lt; rowsOfFirstMatrix; i++) {
         for (int j = 0; j &lt; columnsOfFirstMatrix; j++) {
            aMatrix[i][j] = cmd.nextInt();
         }
      }
```

```
        System.out.println("Enter the number of rows and columns of the
                           second matrix");
        int rowsOfSecondMatrix = cmd.nextInt();
        int columnsOfSecondMatrix = cmd.nextInt();

        // safety net - check order or each matrix, whether eligible for
        // multiplication or not
        while (columnsOfFirstMatrix != rowsOfSecondMatrix) {
           System.out.printf("Matrices with entered orders can't be
                    multiplied with each other, "
                 + "columnsOfFirstMatrix [%d] != rowsOfSecondMatrix [%d]
%n",
                  columnsOfFirstMatrix, rowsOfSecondMatrix);
           System.out.println("Enter the number of rows and columns of
                            second matrix");
           rowsOfSecondMatrix = cmd.nextInt();
           columnsOfSecondMatrix = cmd.nextInt();
        }

        int[][] bMatrix = new
int[rowsOfSecondMatrix][columnsOfSecondMatrix];
        System.out.println("Enter numbers of second matrix");
        for (int i = 0; i &lt; rowsOfSecondMatrix; i++) {
           for (int j = 0; j &lt; columnsOfSecondMatrix; j++) {
              bMatrix[i][j] = cmd.nextInt();
           }
        }

        // calculating product of two matrices in Java
        int[][] product = product(aMatrix, bMatrix);
        System.out.println("Product of entered matrices:-");

        for (int i = 0; i &lt; rowsOfFirstMatrix; i++) {
           for (int j = 0; j &lt; columnsOfSecondMatrix; j++) {
              System.out.printf("%d ", product[i][j]);
           }
           System.out.printf("%n");
        }
        cmd.close();
```

```
    }

    /**
     * Method to calculate multiplication or product of two matrices.
     *
     * @param matrix1
     * @param matrix2
     * @return product of two matrix
     */
    public static int[][] product(int[][] matrix1, int[][] matrix2) {
        int columnsOfFirstMatrix = matrix1[0].length;
        int rowsOfSecondMatrix = matrix2.length;

        if (columnsOfFirstMatrix != rowsOfSecondMatrix) {
            throw new IllegalArgumentException(String.format("Can't
multiply
                    matrices, columns of first matrix"
                  + " %d is not equal to rows of second matrix %d",
                    columnsOfFirstMatrix, rowsOfSecondMatrix));
        }

        int rowsOfFirstMatrix = matrix1.length;
        int columnsofSecondMatrix = matrix2[0].length;
        int[][] product = new
int[rowsOfFirstMatrix][columnsofSecondMatrix];

        for (int i = 0; i &lt; rowsOfFirstMatrix; i++) {
            for (int j = 0; j &lt; columnsofSecondMatrix; j++) {

                int sum = 0;
                for (int k = 0; k &lt; rowsOfSecondMatrix; k++) {
                    sum = sum + matrix1[i][k] * matrix2[k][j];
                }

                product[i][j] = sum;
            }
        }

        return product;
```

```
    }

}
```

Output:

```
Enter the number of rows and columns of the first matrix
2 3
Enter the elements of the first matrix
1 2 3
4 5 6
Enter the number of rows and columns of the second matrix
2 4
Matrices with entered orders can't be multiplied with each other,
columnsOfFirstMatrix [3] != rowsOfSecondMatrix [2]
Enter the number of rows and columns of the second matrix
3 2
Enter numbers of the second matrix
7 8
9 10
11 12
The product of entered matrices:-
58 64
139 154
```

23. Java program to remove white spaces from a string.

```
class RemoveWhiteSpaces
{
   public static void main(String[] args)
   {
      String str = "  Core Java jsp servlets          jdbc struts hibernate spring
";

      //1. Using replaceAll() Method

      String strWithoutSpace = str.replaceAll("\\s", "");
```

```
        System.out.println(strWithoutSpace);        //Output :
CoreJavajspservletsjdbcstrutshibernatespring

        //2. Without Using replaceAll() Method

        char[] strArray = str.toCharArray();

        StringBuffer sb = new StringBuffer();

        for (int i = 0; i < strArray.length; i++)
        {
            if( (strArray[i] != ' ') && (strArray[i] != '\t') )
            {
                sb.append(strArray[i]);
            }
        }

        System.out.println(sb);          //Output :
CoreJavajspservletsjdbcstrutshibernatespring
    }
}
```

24. Java Program To Check Whether Given Number Is An Armstrong Number Or Not :

```
public class MainClass
{
    static void checkArmstrongNumber(int number)
    {
        int copyOfNumber = number;

        int noOfDigits = String.valueOf(number).length();

        int sum = 0;

        while (copyOfNumber != 0)
        {
            int lastDigit = copyOfNumber % 10;
```

```
            int lastDigitToThePowerOfNoOfDigits = 1;

            for(int i = 0; i < noOfDigits; i++)
            {
               lastDigitToThePowerOfNoOfDigits =
lastDigitToThePowerOfNoOfDigits * lastDigit;
            }

            sum = sum + lastDigitToThePowerOfNoOfDigits;

            copyOfNumber = copyOfNumber / 10;
         }

         if (sum == number)
         {
            System.out.println(number+" is an armstrong number");
         }
         else
         {
            System.out.println(number+" is not an armstrong number");
         }
      }

      public static void main(String[] args)
      {
         checkArmstrongNumber(153);

         checkArmstrongNumber(371);

         checkArmstrongNumber(9474);

         checkArmstrongNumber(54748);

         checkArmstrongNumber(407);

         checkArmstrongNumber(1674);
      }
}
```

Output :

153 is an armstrong number
371 is an armstrong number
9474 is an armstrong number
54748 is an armstrong number
407 is an armstrong number
1674 is not an armstrong number

25. Java Program To Find The Sum Of All Digits Of A Number In Java :

```
public class MainClass
{
   static void sumOfAllDigits(int inputNumber)
   {
      //Creating a copy of input number

      int copyOfInputNumber = inputNumber;

      //Initializing sum to 0

      int sum = 0;

      while (copyOfInputNumber != 0)
      {
         //Getting last digit of the input number

         int lastDigit = copyOfInputNumber%10;

         //Adding last digit to sum

         sum = sum + lastDigit;

         //Removing last digit from the input number

         copyOfInputNumber = copyOfInputNumber/10;
      }

      //Printing sum
```

```
        System.out.println("Sum Of All Digits In "+inputNumber+" = "+sum);
    }

    public static void main(String[] args)
    {
        sumOfAllDigits(47862);

        sumOfAllDigits(416872);

        sumOfAllDigits(5674283);

        sumOfAllDigits(475496215);
    }
}
```

Output :

```
Sum Of All Digits In 47862 = 27
Sum Of All Digits In 416872 = 28
Sum Of All Digits In 5674283 = 35
Sum Of All Digits In 475496215 = 43
```

26. Java Program To Find Second Largest Number In An Integer Array :

```
public class MainClass
{
    static int secondLargest(int[] input)
    {
        int firstLargest, secondLargest;

        //Checking first two elements of input array

        if(input[0] > input[1])
        {
            //If first element is greater than second element

            firstLargest = input[0];

            secondLargest = input[1];
```

```
        }
        else
        {
            //If second element is greater than first element

            firstLargest = input[1];

            secondLargest = input[0];
        }

        //Checking remaining elements of input array

        for (int i = 2; i < input.length; i++)
        {
            if(input[i] > firstLargest)
            {
                //If element at 'i' is greater than 'firstLargest'

                secondLargest = firstLargest;

                firstLargest = input[i];
            }
            else if (input[i] < firstLargest && input[i] > secondLargest)
            {
                //If element at 'i' is smaller than 'firstLargest' and greater than
'secondLargest'

                secondLargest = input[i];
            }
        }

        return secondLargest;
    }

    public static void main(String[] args)
    {
        System.out.println(secondLargest(new int[] {45, 51, 28, 75, 49, 42}));
```

```
        System.out.println(secondLargest(new int[] {985, 521, 975, 831, 479,
861}));

        System.out.println(secondLargest(new int[] {9459, 9575, 5692, 1305,
1942, 9012}));

        System.out.println(secondLargest(new int[] {47498, 14526, 74562,
42681, 75283, 45796}));
    }
}
```

Output :

```
51
975
9459
74562
```

27. Java Program To Count Occurrences Of Each Character In String :

```
class EachCharCountInString
{
    static void characterCount(String inputString)
    {
        //Creating a HashMap containing char as a key and occurrences as a
value

        HashMap<Character, Integer> charCountMap = new
HashMap<Character, Integer>();

        //Converting given string to char array

        char[] strArray = inputString.toCharArray();

        //checking each char of strArray

        for (char c : strArray)
        {
            if(charCountMap.containsKey(c))
            {
```

```
            //If char is present in charCountMap, incrementing it's count by
1

            charCountMap.put(c, charCountMap.get(c)+1);
         }
         else
         {
            //If char is not present in charCountMap,
            //putting this char to charCountMap with 1 as it's value

            charCountMap.put(c, 1);
         }
      }

      //Printing the charCountMap

      System.out.println(charCountMap);
   }

   public static void main(String[] args)
   {
      characterCount("Java J2EE Java JSP J2EE");

      characterCount("All Is Well");

      characterCount("Done And Gone");
   }
}
```

Output :

```
{E=4, 2=2, v=2, =4, P=1, S=1, a=4, J=5}
{W=1, =2, e=1, s=1, A=1, l=4, I=1}
{D=1, d=1, =2, G=1, e=2, A=1, n=3, o=2}
```

28. Java Program To Find All Pairs Of Elements In An Array Whose Sum Is Equal To A Given Number :

```
public class PairsOfElementsInArray
{
```

```
   static void findThePairs(int inputArray[], int inputNumber)
   {
      System.out.println("Pairs of elements whose sum is
"+inputNumber+" are : ");

      for (int i = 0; i < inputArray.length; i++)
      {
         for (int j = i+1; j < inputArray.length; j++)
         {
            if(inputArray[i]+inputArray[j] == inputNumber)
            {
               System.out.println(inputArray[i]+" + "+inputArray[j]+" =
"+inputNumber);
            }
         }
      }
   }

   public static void main(String[] args)
   {
      findThePairs(new int[] {4, 6, 5, -10, 8, 5, 20}, 10);

      findThePairs(new int[] {4, -5, 9, 11, 25, 13, 12, 8}, 20);

      findThePairs(new int[] {12, 13, 40, 15, 8, 10, -15}, 25);

      findThePairs(new int[] {12, 23, 125, 41, -75, 38, 27, 11}, 50);
   }
}
```

Output :

```
Pairs of elements whose sum is 10 are :
4 + 6 = 10
5 + 5 = 10
-10 + 20 = 10
Pairs of elements whose sum is 20 are :
-5 + 25 = 20
9 + 11 = 20
12 + 8 = 20
```

Pairs of elements whose sum is 25 are :
12 + 13 = 25
40 + -15 = 25
15 + 10 = 25
Pairs of elements whose sum is 50 are :
12 + 38 = 50
23 + 27 = 50
125 + -75 = 50

29. Separate Zeros From Non-Zeros In An Array

```
public class SeparateZerosFromNonZeros
{
   static void moveZerosToEnd(int inputArray[])
   {
      //Initializing counter to 0

      int counter = 0;

      //Traversing inputArray from left to right

      for (int i = 0; i < inputArray.length; i++)
      {
         //If inputArray[i] is non-zero

         if(inputArray[i] != 0)
         {
            //Assigning inputArray[i] to inputArray[counter]

            inputArray[counter] = inputArray[i];

            //Incrementing the counter by 1

            counter++;
         }
      }

      //Assigning zero to remaining elements
```

```
        while (counter < inputArray.length)
        {
            inputArray[counter] = 0;

            counter++;
        }

        System.out.println(Arrays.toString(inputArray));
    }

    public static void main(String[] args)
    {
        moveZerosToEnd(new int[] {12, 0, 7, 0, 8, 0, 3});

        moveZerosToEnd(new int[] {1, -5, 0, 0, 8, 0, 1});

        moveZerosToEnd(new int[] {0, 1, 0, 1, -5, 0, 4});

        moveZerosToEnd(new int[] {-4, 1, 0, 0, 2, 21, 4});
    }
}
```

Output :

```
[12, 7, 8, 3, 0, 0, 0]
[1, -5, 8, 1, 0, 0, 0]
[1, 1, -5, 4, 0, 0, 0]
[-4, 1, 2, 21, 4, 0, 0]
```

30. Java Program To Reverse The String With Preserving The Position Of Spaces :

```
public class MainClass
{
    static void reverseString(String inputString)
    {
        //Converting inputString to char array 'inputStringArray'
```

```
        char[] inputStringArray = inputString.toCharArray();

        //Defining a new char array 'resultArray' with same size as
inputStringArray

        char[] resultArray = new char[inputStringArray.length];

        //First for loop :
        //For every space in the 'inputStringArray',
        //we insert spaces in the 'resultArray' at the corresponding positions

        for (int i = 0; i < inputStringArray.length; i++)
        {
           if (inputStringArray[i] == ' ')
           {
              resultArray[i] = ' ';
           }
        }

        //Initializing 'j' with length of resultArray

        int j = resultArray.length-1;

        //Second for loop :
        //we copy every non-space character of inputStringArray
        //from first to last at 'j' position of resultArray

        for (int i = 0; i < inputStringArray.length; i++)
        {
           if (inputStringArray[i] != ' ')
           {
              //If resultArray already has space at index j then decrementing 'j'

              if(resultArray[j] == ' ')
              {
                 j--;
              }

              resultArray[j] = inputStringArray[i];
```

```
            j--;
          }
       }

       System.out.println(inputString+" ---> "+String.valueOf(resultArray));
   }

   public static void main(String[] args)
   {
       reverseString("I Am Not String");

       reverseString("JAVA JSP ANDROID");

       reverseString("1 22 333 4444 55555");
   }
}
```

Output :

I Am Not String —> g ni rtS toNmAI
JAVA JSP ANDROID —> DIOR DNA PSJAVAJ
1 22 333 4444 55555 —> 5 55 554 4443 33221

31. Java Program To Find The Percentage Of Uppercase Letters, Lowercase Letters, Digits And Other Special Characters In A String :

```
import java.text.DecimalFormat;

public class MainClass
{
   static void characterPercentage(String inputString)
   {
       //Getting total no of characters in the given string

       int totalChars = inputString.length();

       //Initializing upperCaseLetters, lowerCaseLetters, digits and others
with 0
```

```
int upperCaseLetters = 0;

int lowerCaseLetters = 0;

int digits = 0;

int others = 0;

//Iterating through each character of inputString

for (int i = 0; i < inputString.length(); i++)
{
   char ch = inputString.charAt(i);

   //If ch is in uppercase, then incrementing upperCaseLetters

   if(Character.isUpperCase(ch))
   {
      upperCaseLetters++;
   }

   //If ch is in lowercase, then incrementing lowerCaseLetters

   else if(Character.isLowerCase(ch))
   {
      lowerCaseLetters++;
   }

   //If ch is a digit, then incrementing digits

   else if (Character.isDigit(ch))
   {
      digits++;
   }

   //If ch is a special character then incrementing others

   else
```

```
        {
            others++;
        }
    }

    //Calculating percentage of uppercase letters, lowercase letters, digits
and other characters

    double upperCaseLetterPercentage = (upperCaseLetters * 100.0) /
totalChars ;

    double lowerCaseLetterPercentage = (lowerCaseLetters * 100.0) /
totalChars;

    double digitsPercentage = (digits * 100.0) / totalChars;

    double otherCharPercentage = (others * 100.0) / totalChars;

    DecimalFormat formatter = new DecimalFormat("##.##");

    //Printing percentage of uppercase letters, lowercase letters, digits
and other characters

    System.out.println("In '"+inputString+"' : ");

    System.out.println("Uppercase letters are
"+formatter.format(upperCaseLetterPercentage)+"% ");

    System.out.println("Lowercase letters are
"+formatter.format(lowerCaseLetterPercentage)+"%");

    System.out.println("Digits Are
"+formatter.format(digitsPercentage)+"%");

    System.out.println("Other Characters Are
"+formatter.format(otherCharPercentage)+"%");

    System.out.println("-----------------------------------------");
  }
```

```
    public static void main(String[] args)
    {
        characterPercentage("Tiger Runs @ The Speed Of 100 km/hour.");

        characterPercentage("My e-mail : eMail_Address321@anymail.com");

        characterPercentage("AUS : 123/3, 21.2 Overs");
    }
}
```

Output :

READ : How To Launch External Applications Through Java Program

In 'Tiger Runs @ The Speed Of 100 km/hour.' :
Uppercase letters are 13.16%
Lowercase letters are 52.63%
Digits Are 7.89%
Other Characters Are 26.32%
————————————–
In 'My e-mail : eMail_Address321@anymail.com' :
Uppercase letters are 7.5%
Lowercase letters are 65%
Digits Are 7.5%
Other Characters Are 20%
————————————–
In 'AUS : 123/3, 21.2 Overs' :
Uppercase letters are 17.39%
Lowercase letters are 17.39%
Digits Are 30.43%
Other Characters Are 34.78%

32. Generating Random Numbers In The Given Range :

```
public class MainClass
{
    public static void main(String[] args)
    {
        //Generating random integers between 0 and 50 using Random class
```

```
        System.out.println("Random integers between 0 and 50 using
Random class :");

        Random random = new Random();

        for (int i = 0; i < 5; i++)
        {
            System.out.println(random.nextInt(50));
        }

        //Generating random integers between 0 and 50 range using
Math.random()

        System.out.println("Random integers between 0 and 50 using
Math.random() :");

        for (int i = 0; i < 5; i++)
        {
            System.out.println((int)(Math.random() * 50));
        }

        //Generating random integers between 0 and 50 using
ThreadLoclaRandom

        System.out.println("Random integers between 0 and 50 using
ThreadLocalRandom :");

        for (int i = 0; i < 5; i++)
        {
            System.out.println(ThreadLocalRandom.current().nextInt(0, 50));
        }
    }
}
```

Output :

```
Random integers between 0 and 50 using Random class :
16
12
30
26
```

```
17
Random integers between 0 and 50 using Math.random() :
12
43
42
32
45
Random integers between 0 and 50 using ThreadLocalRandom
12
40
16
17
3
```

33. Java Program To Find Longest Substring Without Repeating Characters :

```
import java.util.LinkedHashMap;

public class MainClass
{
   static void longestSubstring(String inputString)
   {
      //Convert inputString to charArray

      char[] charArray = inputString.toCharArray();

      //Initialization

      String longestSubstring = null;

      int longestSubstringLength = 0;

      //Creating LinkedHashMap with characters as keys and their position
as values.

      LinkedHashMap<Character, Integer> charPosMap = new
LinkedHashMap<Character, Integer>();
```

```
        //Iterating through charArray

        for (int i = 0; i < charArray.length; i++)
        {
            char ch = charArray[i];

            //If ch is not present in charPosMap, adding ch into charPosMap
along with its position

            if(!charPosMap.containsKey(ch))
            {
                charPosMap.put(ch, i);
            }

            //If ch is already present in charPosMap, reposioning the cursor i
to the position of ch and clearing the charPosMap

            else
            {
                i = charPosMap.get(ch);

                charPosMap.clear();
            }

            //Updating longestSubstring and longestSubstringLength

            if(charPosMap.size() > longestSubstringLength)
            {
                longestSubstringLength = charPosMap.size();

                longestSubstring = charPosMap.keySet().toString();
            }
        }

        System.out.println("Input String : "+inputString);

        System.out.println("The longest substring : "+longestSubstring);
```

```
        System.out.println("The longest Substring Length :
"+longestSubstringLength);
    }

    public static void main(String[] args)
    {
        longestSubstring("javaconceptoftheday");

        System.out.println("=========================");

        longestSubstring("thelongestsubstring");
    }
}
```

Output :

READ : How To Reverse The String With Preserving The Position Of Spaces?

```
Input String : javaconceptoftheday
The longest substring : [o, f, t, h, e, d, a, y]
The longest Substring Length : 8
=========================
Input String : thelongestsubstring
The longest substring : [u, b, s, t, r, i, n, g]
The longest Substring Length : 8
```

34. Java Program To Swap Two String Variables Without Using Third Variable :

```
import java.util.Scanner;

public class SwapTwoStrings
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter First String :");

        String s1 = sc.next();
```

```
        System.out.println("Enter Second String :");
        String s2 = sc.next();
        System.out.println("Before Swapping :");
        System.out.println("s1 : "+s1);
        System.out.println("s2 : "+s2);
        //Swapping starts
        s1 = s1 + s2;
        s2 = s1.substring(0, s1.length()-s2.length());
        s1 = s1.substring(s2.length());
        //Swapping ends
        System.out.println("After Swapping :");
        System.out.println("s1 : "+s1);
        System.out.println("s2 : "+s2);
    }
}
```

Output :

```
Enter First String :
JAVA
Enter Second String :
J2EE
Before Swapping :
s1 : JAVA
s2 : J2EE
After Swapping :
s1 : J2EE
```

35. Java Program To Check If Number Belongs To Fibonacci Series Or Not

```
import java.util.Scanner;
public class FibonacciSeries
{
   public static void main(String[] args)
   {
      Scanner sc = new Scanner(System.in);

      System.out.println("Enter positive number :");

      int inputNumber = sc.nextInt();

      int firstTerm = 0;

      int secondTerm = 1;

      int thirdTerm = 0;

      while (thirdTerm < inputNumber)
      {
         thirdTerm = firstTerm + secondTerm;

         firstTerm = secondTerm;

         secondTerm = thirdTerm;
      }

      if(thirdTerm == inputNumber)
      {
         System.out.println("Number belongs to Fibonacci series");
      }
      else
      {
         System.out.println("Number doesn't belongs to Fibonacci series");
      }
   }
```

```
}
```

Output :

1)

```
Enter positive number :
4123
Number doesn't belongs to Fibonacci series
2)
Enter positive number :
17711
Number belongs to Fibonacci series
```

36. Java Program To Print Floyd's Triangle :

```
import java.util.Scanner;

public class FloydsTriangle
{
   public static void main(String[] args)
   {
      System.out.println("How many rows you want in Floyd's Triangle?");

      Scanner sc = new Scanner(System.in);

      int noOfRows = sc.nextInt();

      int value = 1;

      System.out.println("Floyd's Triangle : ");

      for (int i = 1; i <= noOfRows; i++)
      {
         for (int j = 1; j <= i; j++)
         {
            System.out.print(value+"\t");

            value++;
```

```
            }

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

Output :

```
How many rows you want in Floyd's Triangle?
5
Floyd's Triangle :
1
2  3
4  5  6
7  8  9  10
11  12  13  14  15
```

37. Java Program To Create Spiral Of Numbers (Spiral Matrix) In Clockwise Direction?

```
import java.util.Scanner;

public class MainClass
{
   public static void main(String args[])
   {
      System.out.println("Enter The Value For N :");
      Scanner sc = new Scanner(System.in);
      int n = sc.nextInt();
      int[][] spiral = new int[n][n];
      int value = 1;
      int minCol = 0;
      int maxCol = n-1;
      int minRow = 0;
      int maxRow = n-1;
      while (value <= n*n)
      {
         for (int i = minCol; i <= maxCol; i++)
         {
```

```
            spiral[minRow][i] = value;
            value++;
        }

        for (int i = minRow+1; i <= maxRow; i++)
        {
            spiral[i][maxCol] = value;
            value++;
        }

        for (int i = maxCol-1; i >= minCol; i--)
        {
            spiral[maxRow][i] = value;
            value++;
        }

        for (int i = maxRow-1; i >= minRow+1; i--)
        {
            spiral[i][minCol] = value;
            value++;
        }

        minCol++;
        minRow++;
        maxCol--;
        maxRow--;
    }

    for (int i = 0; i < spiral.length; i++)
    {
        for (int j = 0; j < spiral.length; j++)
        {
            System.out.print(spiral[i][j]+ "\t");
        }

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

Output :

Enter The Value For N :
5

```
1     2     3     4     5
16    17    18    19    6
15    24    25    20    7
14    23    22    21    8
13    12    11    10    9
```

38. Java Program To Find The Most Repeated Word In Text File :

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map.Entry;
import java.util.Set;

public class RepeatedWordInFile
{
   public static void main(String[] args)
   {
      //Creating wordCountMap which holds words as keys and their
occurrences as values

      HashMap<String, Integer> wordCountMap = new HashMap<String,
Integer>();

      BufferedReader reader = null;

      try
      {
         //Creating BufferedReader object
```

```
        reader = new BufferedReader(new FileReader("C:\\sample.txt"));

        //Reading the first line into currentLine

        String currentLine = reader.readLine();

        while (currentLine != null)
        {
           //splitting the currentLine into words

           String[] words = currentLine.toLowerCase().split(" ");

           //Iterating each  word

           for (String word : words)
           {
              //if word is already present in wordCountMap, updating its
count

              if(wordCountMap.containsKey(word))
              {
                 wordCountMap.put(word, wordCountMap.get(word)+1);
              }

              //otherwise inserting the word as key and 1 as its value
              else
              {
                 wordCountMap.put(word, 1);
              }
           }

           //Reading next line into currentLine

           currentLine = reader.readLine();
        }

        //Getting the most repeated word and its occurrence

        String mostRepeatedWord = null;
```

```
        int count = 0;

        Set<Entry<String, Integer>> entrySet = wordCountMap.entrySet();

        for (Entry<String, Integer> entry : entrySet)
        {
           if(entry.getValue() > count)
           {
              mostRepeatedWord = entry.getKey();

              count = entry.getValue();
           }
        }

        System.out.println("The most repeated word in input file is :
"+mostRepeatedWord);

        System.out.println("Number Of Occurrences : "+count);
     }
     catch (IOException e)
     {
        e.printStackTrace();
     }
     finally
     {
        try
        {
           reader.close();          //Closing the reader
        }
        catch (IOException e)
        {
           e.printStackTrace();
        }
     }
  }
}
```

Input File :

Java JDBC JSP Servlets
Struts Hibernate java Web Services
Spring JSF JAVA
Threads JaVa Concurrent Programming
jAvA Hadoop Jdbc jsf
spring Jsf jdbc hibernate

Output :

The most repeated word in input file is : java
Number Of Occurrences : 5

JDBC

1. What is JDBC?

JDBC technology is an API that provides connectivity to a wide range of SQL databases and access to other tabular data sources, such as spreadsheets or flat files.

Or

JDBC is java database connectivity as name implies it's a java API for communicating to relational database

This API has java classes and interfaces using which developer can easily interact with database

2. What is JDBC Driver?

The JDBC Driver provides vendor-specific implementations of the abstract classes and interfaces provided by the JDBC API. This driver is used to connect to the database.

Or

JDBC driver is used to establish a connection with the database so that you can fetch, update and maintain database tables using SQL queries.

Or

JDBC Driver is a software component that enables java application to interact with the database.

3. What are the different types of JDBC drivers available?

1. JDBC-ODBC Bridge Driver (Type 1): It uses ODBC driver to connect to database. We should have ODBC drivers installed to connect to database, that's why this driver is almost obsolete.
2. Native driver (Type 2): This driver converts JDBC class to the client API for the database servers. We should have database client API installed. Because of extra dependency on database client API drivers, this is also not preferred driver.
3. Network Protocol Driver (Type 3): This driver sends the JDBC calls to a middleware server that can connect to different type of databases. We should have a middleware server installed to work with this driver. This adds to extra network calls and slow performance and hence it is not widely used JDBC driver.
4. Thin Driver (Type 4): This driver converts the JDBC calls to the vendor specific protocols that are understood by the database

server. This solution is simple and suitable for database connectivity over the network. However for this solution, we should use database specific drivers, for example OJDBC jars by Oracle for Oracle DB and MySQL Connector/J for MySQL databases..

Or

1) JDBC-ODBC bridge driver
The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

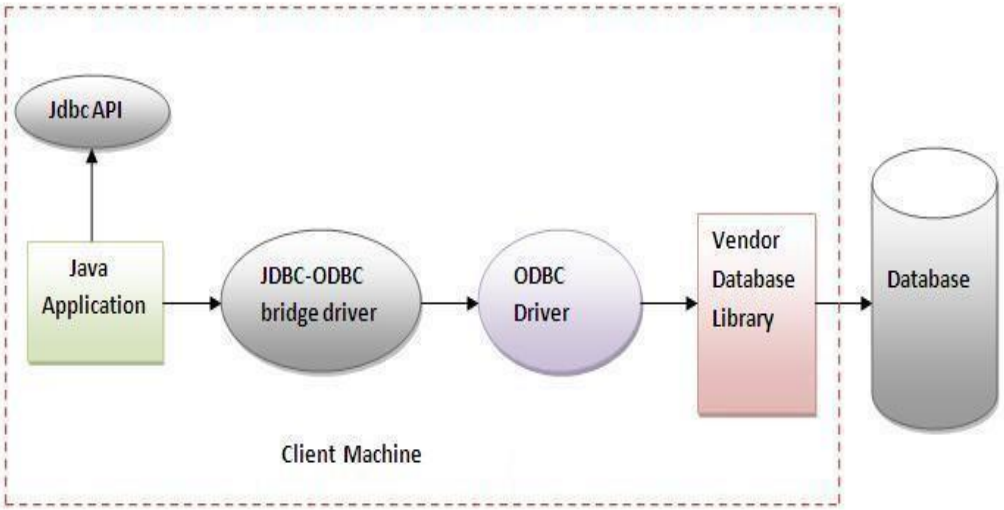


Figure- JDBC-ODBC Bridge Driver

Advantages:

1. easy to use.
2. can be easily connected to any database.

Disadvantages:

1. Performance degraded because JDBC method call is converted into the ODBC function calls.
2. The ODBC driver needs to be installed on the client machine.

2) Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

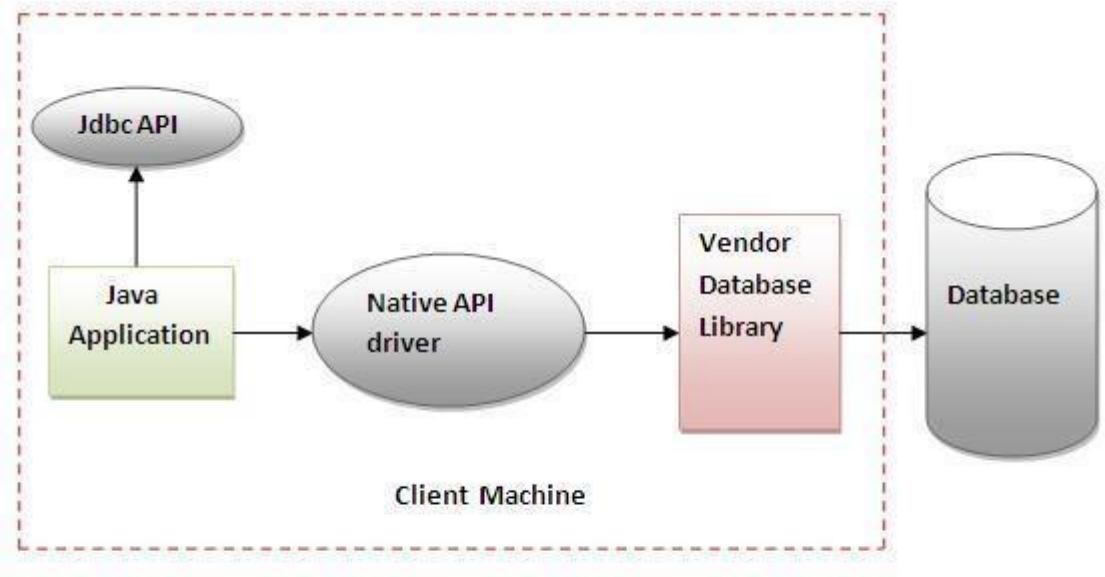


Figure- Native API Driver

Advantage:

1. performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

1. The Native driver needs to be installed on the each client machine.
2. The Vendor client library needs to be installed on client machine.

3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

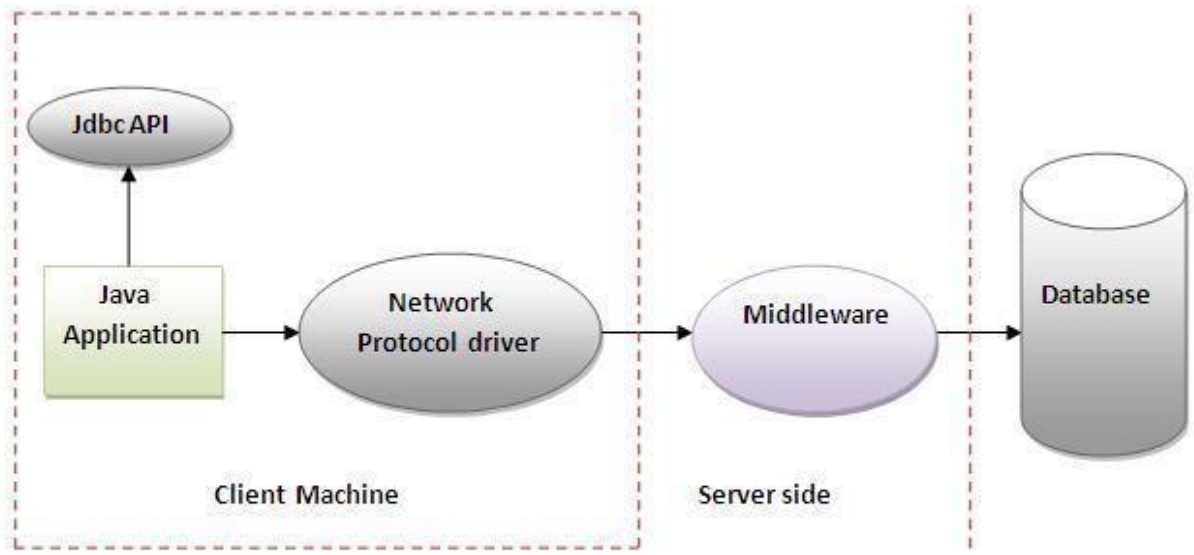


Figure- Network Protocol Driver

Advantage:

No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

Network support is required on client machine.

Requires database-specific coding to be done in the middle tier.

Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4) Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

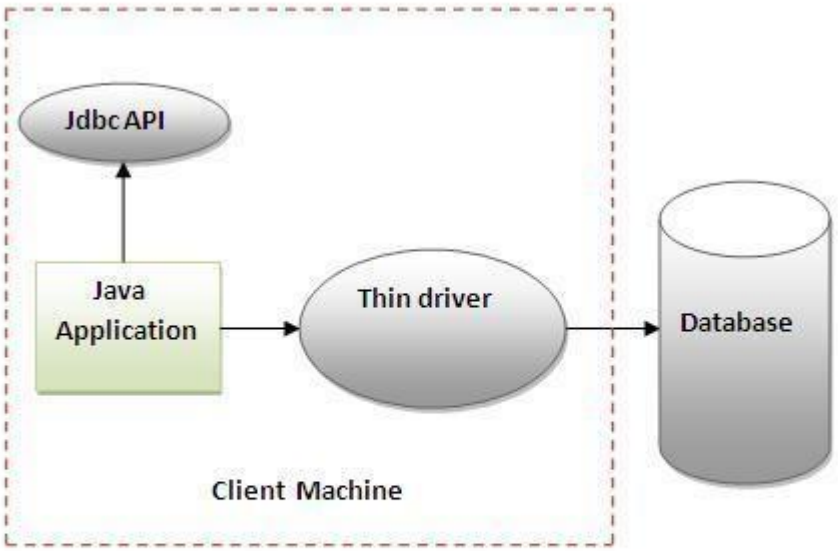


Figure- Thin Driver

Advantage:

Better performance than all other drivers.

No software is required at client side or server side.

Disadvantage:

Drivers depend on the Database.

4).Which is the fastest type of JDBC driver?

Type 4 (JDBC Net pure Java Driver) is the fastest JDBC driver. Type 1 and Type 3 drivers will be slower than Type 2 drivers and Type 4 drivers are the fastest.

5).What are the steps required to execute a query in JDBC?

i. First we need to create an instance of a JDBC driver or load JDBC drivers.
ii. we should open a connection or we should establish the connection
iii. We should create a statement object and this object will help us to execute the query.
iv. We should Execute query
v. We should Display the result
vi. We should Close the resources

Or

- Load the Driver: First step is to load the database specific driver which communicates with database.
- Establish the Connection: Next step is get connection from the database using connection object, which is used to send SQL statement also and get result back from the database.
- Create Statement object: From connection object we can get statement object which is used to query the database
- Execute the Query: Using statement object we execute the SQL or database query and get result set from the query.
- Display the result: the result obtained from Database should be displayed on the screen
- Close the Resources: After getting ResultSet and all required operation performed the last step should be closing the Resources.

6).What Class.forName ( ) method will do?

Class.forName() loads the class dynamically and it returns the object of type class.

Example:we can use Class.forName() to load the driver class in JDBC

As shown below

Class.forName("oracle.jdbc.driver.OracleDriver");

7).What is Connection?

Connection represents a connection with a specific database. SQL statements are executed and results are returned within the context of a connection.
We can establish the connection to Database by using getConnection() method present in DriverManager Class as shown in the example below

```
String url="jdbc:oracle:thin:@//localhost:1521/XE";
String user="system";
String password="system";
Connection
con=DriverManager.getConnection(url,user,password);
```

8).What is a database URL or What is JDBC URL ?

A database URL (or JDBC URL) is a platform independent way of addressing a database.

A database/JDBC URL is of the form

[code language="java"]jdbc:[subprotocol]:[node]/[databaseName][/code]

Example: oracle URL is as shown below

```
jdbc:oracle:thin:@//localhost:1521/XE
```

9).What are the different types of Statements available in JDBC?

There are three types of statement:

1. Statement: it's a commonly used for getting data from database useful when we are using static SQL statement at runtime. it will not accept any parameter.
Example:
Statement stmt = conn.createStatement( );

2. PreparedStatement: when we are using same SQL statement multiple times it is useful and it will accept parameter at runtime.

String SQL = "Update stock SET limit =? WHERE stock Type = ?";

Example:
PreparedStatement pstmt = conn.prepareStatement(SQL);

3. Callable Statement: when we want to access stored procedures then callable statement are useful and they also accept runtime parameter. It is called like this
Example:
CallableStatement cs = con.prepareCall("{call SHOW_SUPPLIERS}");

10).Which are the methods that can be used to execute a Query in JDBC

We can make use of any one of the following methods to execute Query in JDBC

execute(): Returns true if the first object that the query returns is a ResultSet object.

executeQuery(): Returns one ResultSet object.

executeUpdate() : Returns an integer representing the number of rows affected by the SQL statement. We use this method if we are using INSERT, DELETE, or UPDATE SQL statements(DML statements).

11).What is the difference between execute, executeQuery, executeUpdate?

execute() - Executes the any kind of SQL statement. It returns boolean

executeQuery() - This is used generally for reading the content of the Database. The output will be in the form of ResultSet. Generally SELECT statement is used.

executeUpdate() - This is generally used for altering the databases. Generally , INSERT into TABLE, UPDATE TABLE, DELETE from TABLE statements(DML Statements) The output will be in the form of int which denotes the number of rows affected by the query.

12). What is JDBC ResultSet?

JDBC ResultSet is like a table of data representing a database result set, which is usually generated by executing a statement that queries the database. ResultSet object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row.
The next() method moves the cursor to the next row. If there are no more rows, next() method returns false and it can be used in a while loop to iterate through the result set.

13). Is it mandatory to close all ResultSet, Statements and Connections?

Yes. When you are using JDBC in your application you are allocating resources not only in your program but in a database server as well. Failure to properly closed Statements(Ststement and PreparedStatement and CallableStatement), ResultSet and Connections can cause all of the following problems

– You will run out of connections that can be opened or used

– You will not be able to create any new ResultSet

– You will not be able to execute any queries of any kind

– Your program will get very slow

– The database server get very slow

– Your program will crash

– The database server will crash

It is advisable that you should always close all the JDBC resources you obtain in the reverse order that you obtained them to avoid these issues.

14). What is metadata?

Metadata: It is nothing but a data definition of data.
In JDBC, we have two types of metadata.
1) MetaData for ResultSet(ResultSetMetaData): ResultSet is used to fetch data from tables and this MetaData keeps record like how many rows

have been fetched, what are the column names etc.

2) MetaData for Database connections(DataBaseMetaData): It has the record of database connections.

15). Why we need to use BatchUpdates?

Batch updates are used to *reduce the Database workloads*. for example if you are going to insert 120 records in database, if you do it using 120 insert SQL statements, it would definitely consume more amount of time as for each INSERT statement a connection needs to be established between DB and resources. If you perform it through batch process then all 120 statements would be inserted in table in one go, which would save time and improves resource utilization.

16). What is transaction?

A transaction is *atomic unit of Work*. Which means a transaction is a collection of several tasks and if task fails then the effect will be rolled back to the previous state. In simple terms a transaction commits only when all the tasks specified in it executes successfully.

17) How do you handle your own transaction ?

Connection Object has a method called setAutocommit (boolean flag) . For handling our own transaction we can set the parameter to false and begin your transaction .Finally commit the transaction by calling the commit() method.

18) Explain JDBC Savepoint?

A savepoint represents a point that the current transaction can roll back to. Instead of rolling all its changes back, it can choose to roll back only some of them.

19). How to rollback a JDBC transaction

We can use Connection object rollback() method to rollback the transaction. It will rollback all the changes made by the transaction .

20). What are the steps followed to create a batch process?

Typical sequences of steps to use Batch Processing with Statement or PrepareStatement Object are.

Create a Statement or PrepareStatement object using either createStatement() or prepareStatement() methods respectively.

Set auto-commit to false using setAutoCommit().

Add as many as SQL statements you like into batch using addBatch() method on created statement object.

Execute all the SQL statements using executeBatch() method on created statement object.

Finally, commit all the changes using commit() method.

21). Explain the ways in which we can get runtime information about the JDBC Driver?

We have to Use the following DatabaseMetaData methods:

getDriverMajorVersion()

getDriverMinorVersion()

getDriverName()

getDriverVersion()

22). What are JDBC Best Practices?

Some of the JDBC Best Practices are.

i) Database resources are heavy, so make sure you close it as soon as you are done with it. Connection, Statement, ResultSet and all other JDBC objects have close() method defined to close them

ii) Always close the result set, statement and connection explicitly in the code, because if you are working in connection pooling environment, the connection might be returned to the pool leaving open result sets and statement objects resulting in resource leak

iii) Close the resources in the finally block to make sure they are closed even in case of exception scenarios

iv) Use batch processing for bulk operations of similar kind

v) Always use PreparedStatement over Statement to avoid SQL Injection and get pre-compilation and caching benefits of PreparedStatement.

23). What are SQL warnings?

SQLWarning objects are a subclass of SQLException that deal with database access warnings. Warnings do not stop the execution of an application, as exceptions do. They simply alert the user that something did not happen as planned. A warning can be reported on a Connection object, a Statement object (including PreparedStatement and CallableStatement objects), or a ResultSet object. Each of these classes has a getWarnings() method by using which we can get SQLwarnings.

24). What is difference between java.util.Date and java.sql.Date?

java.util.Date contains information about the date and time whereas java.sql.Date contains information only about the date, it doesn't have time information. So if you have to keep time information in the database, it is advisable to use Timestamp or DateTime fields.

25). What do you understand by DDL and DML statements ?

Data Definition Language (DDL) statements are used to define the database schema. Create, Alter, Drop, Truncate, Rename statements comes under DDL statements and usually they don't return any result.

Data Manipulation Language (DML) statements are used to manipulate data in the database schema. Select, Insert, Update, Delete, Call etc are example of DML statements.

26). What are stored procedures?

A stored procedure is a set of statements/commands which reside in the database. The stored procedure is precompiled.

27). Prepared Statements are faster. Why?

Prepared statement execution is faster than direct execution because the statement is compiled only once. Prepared statements & JDBC driver are connected with each other during execution, and there are no connection overheads.

28) How to use JDBC API to call Stored Procedures?

Stored Procedures are group of SQL queries that are compiled in the database and can

be executed from JDBC API. JDBC CallableStatement can be used to execute

stored procedures in the database. The syntax to initialize CallableStatement is

CallableStatement stmt = con.prepareCall("{call insertEmployee(?,?,?,?,?,?)}");

stmt.setInt(1, id);

stmt.setString(2, name);

stmt.setString(3, role);

stmt.setString(4, city);

stmt.setString(5, country);

stmt.registerOutParameter(6, java.sql.Types.VARCHAR);

stmt.executeUpdate();

We need to register the OUT parameters before executing the CallableStatement.

29). Why “No suitable driver” error occurs?

No suitable driver” occurs when we are calling DriverManager.getConnection() method,

it may occur because of following reasons:

1.unable to load exact JDBC drivers before calling the getConnection() method.

2.URL may be invalid or wrong JDBC URL.

30). What is the meaning of “dirty read“in database?

As the name conveys the meaning of dirty read is “reading the value which may or may not be correct”. in database when one transaction is executing and changing some fields value, and at the same time some another transaction comes and reads the changed fields value before first transaction commits or rollbacks the value then it is said to have read a wrong data . This scenario is known as dirty read.

31). What is cold backup and hot backup?

Cold backup means all these files must be backed up at the same time, before the database is restarted. Hot backup (official name is 'online backup') is a backup taken of each tablespace while the database is running and is being accessed by the users.

32).What is JDBC DataSource and what are its benefits

JDBC DataSource is the interface defined in javax.sql package and it is more powerful that DriverManager for database connections. We can use DataSource to create the database connection and Driver implementation classes do the actual work for getting connection. Apart from getting Database connection, DataSource provides some additional features such as:

1. Caching of PreparedStatement for faster processing

2. Connection timeout settings

3. Logging features

4. ResultSet maximum size threshold

5. Connection Pooling in servlet container using JNDI support

33). Explain the difference between RowSet vs. ResultSet in JDBC?

In a ResultSet handle connection to a DB, we cannot make Result as a serialized object. Because of above issue, we cannot pass ResultSet across the network.

RowSet extends the ResultSet interface, so it holds all methods from ResultSet. RowSet is serialized. So, we can pass RowSet from one class to another class because it has no connection with the database.

34) Is it possible to connect to multiple databases? Using single statement can we

Yes, it is possible to connect to multiple databases, at the same time, but it depends on the specific driver.

To update and extract data from the different database we can use the single statement. But we need middleware to deal with multiple databases or a single database.

35) Explain the JDBC Architecture?

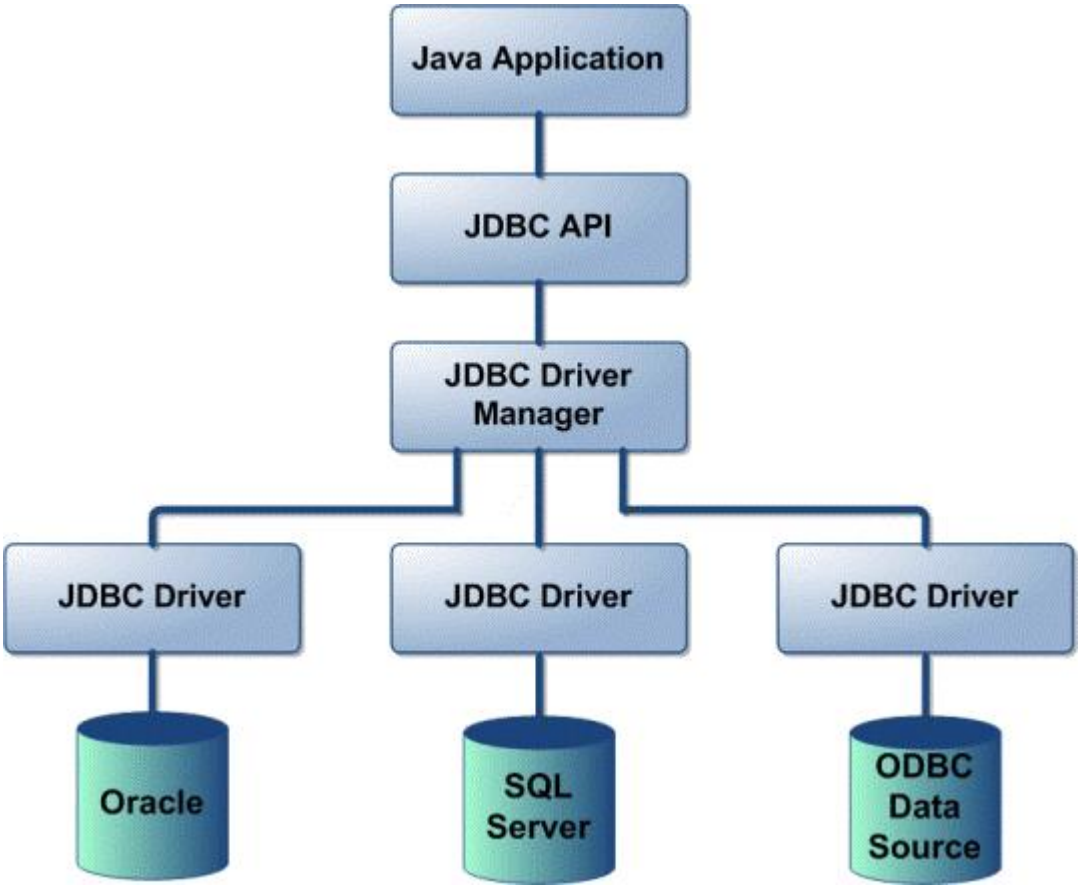
The JDBC Architecture consists of two layers:

The JDBC API, which provides the application-to-JDBC Manager connection.

The JDBC Driver API, which supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases. The JDBC driver manager ensures that the correct driver is used to access each data

source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases. The location of the driver manager with respect to the JDBC drivers and the Java application is shown in Figure below



36).What is connection pooling? what is the main advantage of using connection pooling?

A connection pool is a mechanism to reuse connections created. Connection pooling can increase performance dramatically by reusing connections rather than creating a new physical connection each time a connection is requested..

37).Give an Example code snippet to execute the select Query and display the result in ResultSet.

```
ResultSet rs=stmt.executeQuery("select * from emp");
```

while(rs.next()){

System.out.println(rs.getInt(1)+" "+rs.getString(2)); }

38). Write an Example Snippet to Connect Java Application with mysql database.

```
import java.sql.*;class
MysqlCon
{
    public static void main(String args[])
{
    try
    {
    Class.forName("com.mysql.jdbc.Driver"); Connection
    con=DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/Employees","root","root");
     Statement stmt=con.createStatement();
     ResultSet rs=stmt.executeQuery("select * from emp");
     while(rs.next())
     System.out.println(rs.getInt(1)+" "+rs.getString(2)+"
"+rs.getString(3));
     con.close();
     }
    catch(Exception e)
     {
     System.out.println(e);
      }
}
}
```

39).Mention some of the Methods Present in DriverManager class and its uses

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The

DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

Some of the methods present in DriverManager Class and its uses are shown below

1) public static void registerDriver(Driver driver):
   is used to register the given driver with DriverManager.

2) public static Connection getConnection(String url):
   is used to establish the connection with the specified url.

3) public static Connection getConnection(String url,String userName,String password):
   is used to establish the connection with the specified url, username and password.

4) public static Connection getConnection(String url,Properties prop):
   is used to establish the connection with the specified url and object of properties file.

40)explain with a code snippet How to get the object of ResultSetMetaData

The getMetaData() method of ResultSet interface returns the object of ResultSetMetaData. Syntax:

public ResultSetMetaData getMetaData()throws SQLException

```
import java.sql.*;
class Rsmd{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement ps=con.prepareStatement("select * from emp");
ResultSet rs=ps.executeQuery();
```

```
ResultSetMetaData rsmd=rs.getMetaData();

System.out.println("Total columns: "+rsmd.getColumnCount());
System.out.println("Column Name of 1st column:
"+rsmd.getColumnName(1));
System.out.println("Column Type Name of 1st column:
"+rsmd.getColumnTypeName(1));

con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

41).Write a jdbc program to create a table in the Database.

```
import java.sql.*;


public class JDBCExample
        {

          static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
          static final String URL =
        "jdbc:mysql://localhost/STUDENTS";
          static final String USER = "username";
          static final String PASS = "password";

          public static void main(String[] args)
        {
              Connection con = null;
               Statement stmt = null;
               Try
             {
            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection(URL, USER, PASS);
            stmt = con.createStatement();
```

```
            String sql = "CREATE TABLE REGISTRATION " +
                         "(id INTEGER not NULL, " +
                         " first VARCHAR(255), " +
                         " last VARCHAR(255), " +
                         " age INTEGER, " +
                         " PRIMARY KEY ( id ))";
            stmt.executeUpdate(sql);


          }
        catch(Exception e)
          {
            e.printStackTrace();
          }
        finally
        {
          try
           {
              con.close();
            stmt.close();

           }
           catch(Exception e)
          {
            e.printStackTrace();
          }

        }

        }
}
```

42). Write a jdbc program to insert records in to a table in the Database.

```
import java.sql.*;

public class JDBCExample {

  String URL = "jdbc:mysql://localhost:3306/STUDENTS";
  String USER = "username";
  String PASS = "password";
  public static void main(String[] args)
{
  Connection conn = null;
  Statement stmt = null;
  try
{
    Class.forName("com.mysql.jdbc.Driver");
    conn = DriverManager.getConnection(URL, USER, PASS);
    stmt = conn.createStatement();
    String sql = "INSERT INTO Registration " +
                 "VALUES (100, 'Zara', 'Ali', 18)";
    stmt.executeUpdate(sql);
    sql = "INSERT INTO Registration " +
                 "VALUES (101, 'Mahnaz', 'Fatma', 25)";
    stmt.executeUpdate(sql);
    sql = "INSERT INTO Registration " +
                 "VALUES (102, 'Zaid', 'Khan', 30)";
    stmt.executeUpdate(sql);
    sql = "INSERT INTO Registration " +
                 "VALUES(103, 'Sumit', 'Mittal', 28)";
    stmt.executeUpdate(sql);
  }
catch(Exception e)
{
    e.printStackTrace();
```

```
        }
        finally
        {
          try
           {
              con.close();
              stmt.close();

            }
            catch(Exception e)
          {
            e.printStackTrace();
          }

        }
        }
}
```

43). Write the jdbc program to Delete records in to a table in the Database.

```
public class JDBCExample
{

  String URL = "jdbc:mysql://localhost:3306/STUDENTS";
  String USER = "username";
  String PASS = "password";

  public static void main(String[] args)
  {
  Connection conn = null;
  Statement stmt = null;
  try{
        Class.forName("com.mysql.jdbc.Driver");
    conn = DriverManager.getConnection(URL, USER, PASS);
    stmt = conn.createStatement();
    String sql = "DELETE FROM Registration " +
                 "WHERE id = 101";
    stmt.executeUpdate(sql);
    sql = "SELECT id, first, last, age FROM Registration";
    ResultSet rs = stmt.executeQuery(sql);

    while(rs.next())
    {
      int id  = rs.getInt("id");
      int age = rs.getInt("age");
      String first = rs.getString("first");
      String last = rs.getString("last");
      System.out.print(" ID: " + id);
      System.out.print(" , Age: " + age);
      System.out.print(" , First: " + first);
      System.out.println(" , Last: " + last);
    }
  }
  catch(Exception e)
   {
        e.printStackTrace();
```

```
   }
  finally
{
   try
    {
        con.close();
      stmt.close();
       rs.close();

     }
    catch(Exception e)
   {
     e.printStackTrace();
   }

}
}
}
```

44). Write a jdbc program to Update records in to a table in the Database.

```
import java.sql.*;

public class JDBCExample
{
   static final String DB_URL = "jdbc:mysql://localhost:3306/STUDENTS";
   static final String USER = "username";
   static final String PASS = "password";
   public static void main(String[] args)
 {
   Connection conn = null;
   Statement stmt = null;
   try
{

      Class.forName("com.mysql.jdbc.Driver");
      conn = DriverManager.getConnection(DB_URL, USER, PASS);
      stmt = conn.createStatement();
```

```
    String sql = "UPDATE Registration " +
                 "SET age = 30 WHERE id in (100, 101)";
    stmt.executeUpdate(sql);


    sql = "SELECT id, first, last, age FROM Registration";
    ResultSet rs = stmt.executeQuery(sql);

    while(rs.next())
    {
       int id  = rs.getInt("id");
       int age = rs.getInt("age");
       String first = rs.getString("first");
       String last = rs.getString("last");
       System.out.print("ID: " + id);
       System.out.print(", Age: " + age);
       System.out.print(", First: " + first);
       System.out.println(", Last: " + last);
    }
  }
  catch(Exception e)
   {
    e.printStackTrace();
   }
  finally
{
  try
   {
      con.close();
    stmt.close();
    rs.close();

   }
   catch(Exception e)
  {
    e.printStackTrace();
  }

}
```

```
}
}
```

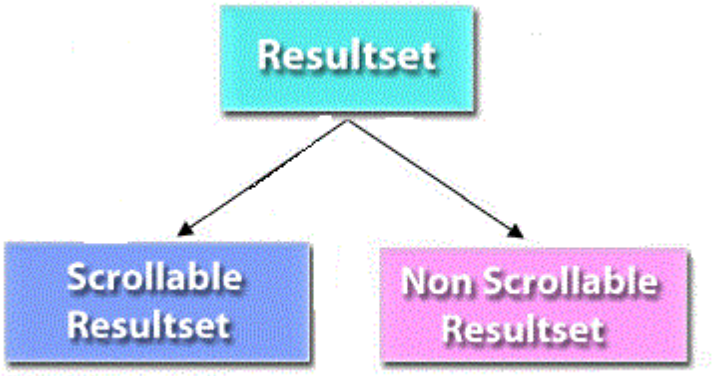
45).Explain Scrollable ResultSet with an Example

In Jdbc ResultSet Interface are classified into two type;.

1.Non-Scrollable ResultSet in JDBC

2.Scrollable ResultSet

By default a ResultSet Interface is Non-Scrollable, In non-scrollable ResultSet we can move only in forward direction (that means from first record to last record), but not in Backward Direction, If you want to move in backward direction use Scrollable Interface.



To create a Scrollable ResultSet, create Statement object with two parameters.

Syntax:

Statement stmt=con.createStatement(param1, param2);

In the above syntax param1 and param2 are modes

These type and mode are predefined in ResultSet Interface of Jdbc like below which is static final.

Types:

public static final int TYPE_FORWARD_ONLY=1003

public static final int TYPE_SCROLL_INSENSITIVE=1004

public static final int TYPE_SCROLL_SENSITIVE=1005

modes:
public static final int CONCUR_READ_ONLY=1007

public static final int CONCUR_UPDATABLE=1008

Note: To create Statement object we can pass either int value as parameter or their variable name.

Statement stmt=con.createStatement(1004, 1007);

Or

Statement stmt=con.createStatement(ResutlSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);

We can use the below mentioned methods to move the cursor in scrollable ResultSet

i)afterLast ():Used to move the cursor after last row.

ii)beforeFirst(): Used to move the cursor before first row.

iii)previous(): Used to move the cursor backward.

iv)first(): Used to move the cursor first at row.

v)last(): Used to move the cursor at last row.

Example:

```
import java.sql.*;
class ScrollableTest
{
public static void main(String[] args) throws Exception
{
Connection con;
Statement  stmt;
ResultSet rs;
try
{
Class.forName("oracle.jdbc.OracleDriver");
```

```
con=DriverManager.getConnection("jdbc:oracle:thin:@//localhost:1521/xe
","system","system");
stmt=con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,Result
Set.CONCUR_READ_ONLY);
rs=stmt.executeQuery("select * from student");

//reading from button to top
rs.afterLast();
while(rs.previous())
{
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getInt(3));
}

//move the cursor to 3rd record
rs.absolute(3);
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getInt(3));

//move the cursor to 2nd record using relative()
rs.relative(-1);
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getInt(3));
int i=rs.getRow(); // get cursor position
System.out.println("cursor position="+i);
}
catch(Exception e)
{
  e.printStackTrace();
}
finally
{
try
{
rs.close();
stmt.close();
con.close();
}
catch(Exception e)
{
  e.printStackTrace();
}
```

```
}
 }
}
```

46). Can I get a null ResultSet ?

No, It is not possible to get null ResultSet. Database Drivers will always return a result set for a query. rs.next() can be used to see whether next record contains a row or not, here rs is a ResultSet object.

JAVA ENTERPRISE EDITION

1).What is a web application?

Web application is a collection of components assembled to provide services over the web for users to consume the service provided. Example: online train ticket reservation, internet banking.

2). How are web application accessed?

Web applications are accessed over the web using URL address. URL stands for Uniform Resource Locator which is a unique identifier available to locate a particular service.

3). How HTTP works?

HTTP stands for Hypertext Transfer Protocol. The following below steps explains about working:

Client opens connection.

Client sends HTTP request.

Server sends HTTP response.

Client closes connection.

4).Why HTTP is called stateless protocol?

Since the connection is open and closed for each HTTP request, the state is not maintained across HTTP requests. Hence HTTP is called "Stateless Protocol".

5). What is a servlet?

A servlet is a java program that runs within a web server and serves the client request .Because it serves the client the name is servlet. Servlets can

receive and respond to requests from web clients working on HTTP protocol.

6). What is a servlet container?

A servlet container is a container for servlets which is responsible for managing the servlets. The main functions are load, initialize and execute servlets to serve client requests. When a request is received, the container decides what servlet to call based on a configuration file.

7). What is CGI?

Common Gateway Interface(CGI) was the most commonly used for server side scripting before the evolution of the java servlet technology. With traditional CGI, a new process is started for each HTTP request eating up more of the CPU cycles thus degrading the performance of the system.

8). What are the advantages of servlet over CGI?

i. When a CGI script is called, a new process is created for each and every client request. But for servlets, each request is handled by a separate thread. Hence, servlet is more efficient because process creation takes more time compared to thread creation.
ii. For each new CGI requests, CGI script is copied into memory. But with servlets, there is only one copy across requests and is shared among threads.

iii. In CGI multiple processes serve concurrent requests. But in servlets, only single Process serves concurrent request.

9). Describe about the life cycle of servlet?

Life cycle methods of a servlet are:

init() method: During initialization web container initializes the servlet instance by calling the init() method passing a ServletConfig interface

object as parameter. The ServletConfig object allows the servlet to access initialization parameters. It is called only once in the life cycle of servlet.

service() method: When client requests are serviced this method gets called. Each requests is serviced in its own thread. For every client requests web container calls the servlet's service() method.

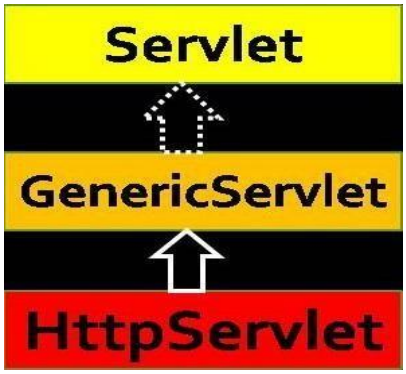
destroy() method: The web container calls the destroy method and takes the servlet out of service. It is called only once in the life cycle of servlet, when you stop the server or when you remove the application from the server.

10).Explain servlet class hierarchy

Interface Servlet: It declares the standard API's which all the servlet implementations needs to define. All servlet classes must implement this interface directly or indirectly.

Class GenericServlet: It defines a generic, protocol independent servlet. Servlet developed to cater NON-HTTP requests.

Class HTTPServlet: Servlet used to cater client HTTP requests.



11). What is init() method of Servlet API?

It is a life cycle method of a servlet .It is called by the servlet container to indicate to a servlet that the servlet is being placed into service. init() method takes ServletConfig object as an argument. ServletConfig object contains initialization and start-up parameters for the servlet.

12). What is destroy() method of Servlet interface?

It is a life cycle method of a servlet .It is invoked by the servlet container to indicate that the servlet is taken out of service.

13). What is service() method of Servlet API?

It is a life cycle method of a servlet .It is invoked by the servlet container to allow the servlet to respond to a request.

14). What is HTTPServlet?

HTTPServlet is an abstract class. Any servlet that works on the HTTP should extend this class

15). Explain briefly the doGet and doPost methods in HTTPServlet?

protected void doGet(HTTPServletRequest req, HTTPServletResponse res)

Above method is called by the container to allow a servlet to handle a GET request. It will be called when a URL is requested or the form method is GET.

protected void doPost(HTTPServletRequest req, HTTPServletResponse res)

It is called by the container to allow a servlet to handle a POST request. It will be called when the form method is POST.

16). What is web.xml file?

web.xml is a configuration file which captures the application configuration and deployment details of a web application.

Hence in web applications we call the web.xml as deployment descriptor

17). What are servlet initialization parameters?

Initialization parameters are used by developers to set some parameter values for the servlet to use. This will be stored as key/value pairs.

18). How do we set init parameter in web.xml?

Init parameters are declared inside <init-param> tag. The parameter name is declared using <param-name> and value using <param-value> . Init parameter is declared separately for each servlet. Each init parameter needs to declared inside a separate <init- param> tag. For example,

```
<servlet>
<display-name>Sample</display-name>
<servlet-name>Sample</servlet-name>
<servlet-class>Sample</servlet-class>

<init-param>
<param-name>URL</param-name>
        <param-
value>jdbc:mysql:@//localhost:3306/Employees</param-value>
</init-param>
<init-param>
        <param-name>USER</param-name>
        <param-value>root</param-value>
</init-param>
        <init-param>
        <param-name>PASSWORD</param-name>
        <param-value>root</param-value>
</init-param>
</servlet>
```

19). How to read initialization parameter?

Init parameter can be read using the getInitParameter(ServletConfig sc) method of ServletConfig object.

String var = sc.getInitParameter(String parameterName)

Where sc is the ServletConfig object and parameterName is the key name of the parameter set in web.xml.

```
Example:
public void init(ServletConfig sc)
        {
                try
                {
                String driver=sc.getInitParameter("DRIVER");
                String url=sc.getInitParameter("USER");
                String password=sc.getInitParameter("PASSWORD");
                }
                catch (Exception e)
                {
                        e.printStackTrace();
                }
        }
```

20). What is a servlet context?

It refers to the context in which the servlet runs. It is a gateway between the web container and the servlets. There will be only one instance of servlet context per web application shared by all the servlets in the application. It is used for storing values which can be shared across all the servlets in the web application.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app >
<context-param>
 <param-name>URL</param-name>
 <param-value>jdbc:mysql:@//localhost:3306/Employees</param-value>
</context-param>
<context-param>
<param-name>USER</param-name>
<param-value>root</param-value>
</context-param>
<context-param>
<param-name>PASSWORD</param-name>
<param-value>root</param-value>
</context-param>
```

```
</web-app>
```

21). How can we read context parameters?

Context parameters can be read using getInitParameter() method of the ServletContext interface.

```
Example:
public class Sample extends HttpServlet
{
        ServletConfig sc;
        ServletContext sct;
        public void init(ServletConfig sc)
        {
                try
                {
                sct=sc.getServletContext();
                String driver=sct.getInitParameter("DRIVER");
                String url=sct.getInitParameter("USER");
                String password=sct.getInitParameter("PASSWORD");
                }
                catch (Exception e)
                {
                        e.printStackTrace();
                }
        }
```

22). What is servlet chaining?

Servlet chaining is a technique in which two or more servlets orchestrate in servicing a single request.

Or

Dispatching the request from one servlet to another servlet is called as servlet chaining

In servlet chaining, one servlet's response is piped to next servlet's input. This process continues until the last servlet is reached. Its output is sent

back to the client. The same request and response object are available across all the servlets in the process.

Example 1: **One level servlet chaining**

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class Servlet1 extends HttpServlet {
        public void service(HttpServletRequest req,HttpServletResponse
res)
        {
                try
                {
        req. getRequestDispatcher("/CHAINING/Servlet2").forward(req,
res);

                }
                catch (Exception e)
                {
                        e.printStackTrace();
                }
        }

}
```

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
public class Servlet2 extends HttpServlet
{
	public void service(HttpServletRequest req , HttpServletResponse
res)
	{
		try
		{
			PrintWriter pw=res.getWriter();
			pw.println("HELLO..");

		}
		catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

Example 2: **Two level servlet chaining**

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class Servlet1 extends HttpServlet {
	public void service(HttpServletRequest req,HttpServletResponse
res)
	{
		try
		{
	req. getRequestDispatcher("/CHAINING/Servlet2").forward(req,
res);

		}
		catch (Exception e)
		{
			e.printStackTrace();
		}
	}
```

```
}
```

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class Servlet2 extends HttpServlet {
	public void service(HttpServletRequest req,HttpServletResponse
res)
	{
		try
		{
	req. getRequestDispatcher("/CHAINING/Servlet3").forward(req,
res);

		}
		catch (Exception e)
		{
			e.printStackTrace();
		}
	}

}
```

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class Servlet2 extends HttpServlet
```

```
{
	public void service(HttpServletRequest req,HttpServletResponse
res)
	{
		try
		{
			PrintWriter pw=res.getWriter();
			pw.println("HELLO..");

		}
		catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

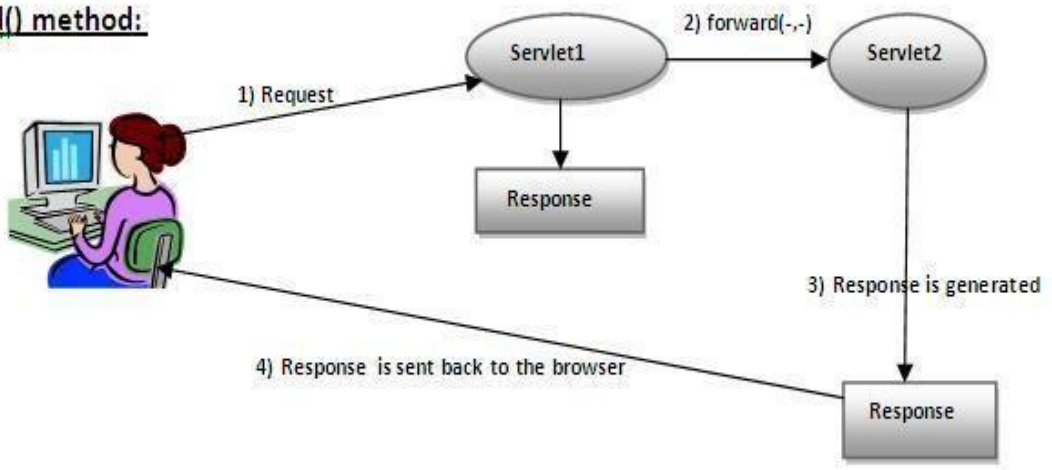
23).Explain about two ways of servlet chaining?

Two ways of servlet chaining are:

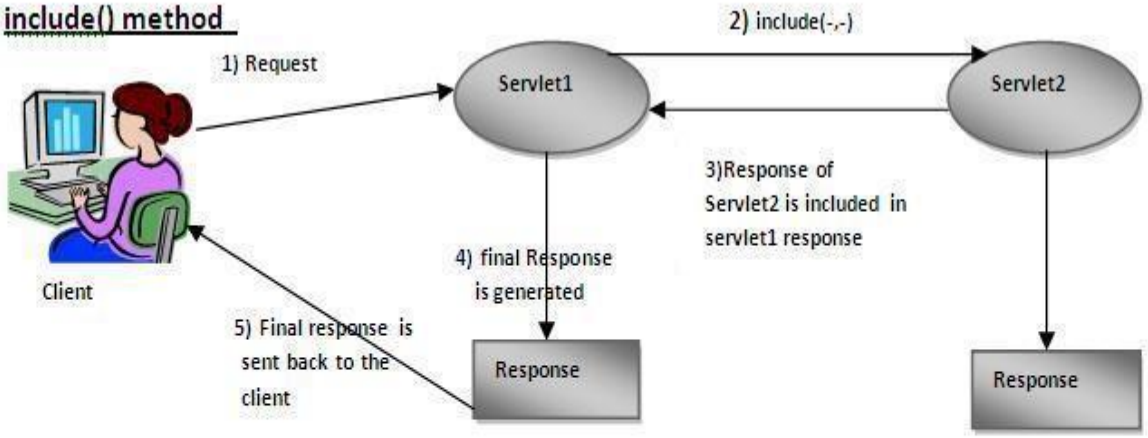
Forward: The servlet being requested by the client forwards the request to one or more servlet and finally the response is sent to the client by last servlet invoked in the Chain.

Include: This refers to the process of including the response of one or more servlets with the response of the servlet being requested and the collated response sent to the client by the servlet being requested.

forward() method:



include() method



Example 1: **Servlet Chaining using forward**

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class Servlet1 extends HttpServlet {
	public void service(HttpServletRequest req,HttpServletResponse
res)
	{
		try
		{
	req. getRequestDispatcher("/CHAINING/Servlet2").forward(req,
res);

		}
		catch (Exception e)
		{
			e.printStackTrace();
		}
	}

}


import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class Servlet2 extends HttpServlet {
	public void service(HttpServletRequest req,HttpServletResponse
res)
	{
		try
```

```
		{

	req.getRequestDispatcher("/CHAINING/Servlet3").forward(req
, res);

		}
		catch (Exception e)
		{
			e.printStackTrace();
		}
	}

}

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class Servlet2 extends HttpServlet
{
	public void service(HttpServletRequest req,HttpServletResponse
res)
	{
		try
		{
			PrintWriter pw=res.getWriter();
			pw.println("HELLO..");

		}
		catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

Example 2: **ServletChaining using include**

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class Servlet1 extends HttpServlet {
	public void service(HttpServletRequest req,HttpServletResponse
res)
	{
		try
		{
req.getRequestDispatcher("/CHAINING/Servlet2").forward(req,res);

		}
		catch (Exception e)
		{
			e.printStackTrace();
		}
	}

}


import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class Servlet2 extends HttpServlet {
	public void service(HttpServletRequest req,HttpServletResponse
res)
	{
		try
		{
```

```
                PrintWriter pw=res.getWriter();
                pw.println("HELLO  ");
req.getServletContext().getRequestDispatcher("/CHAINING/Servlet
3").include(req,res);

                }
                catch (Exception e)
                {
                        e.printStackTrace();
                }
        }

}

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class Servlet2 extends HttpServlet
{
        public void service(HttpServletRequest req,HttpServletResponse
res)
        {
                try
                {
                        PrintWriter pw=res.getWriter();
                        pw.println("World..");

                }
                catch (Exception e)
              {
                        e.printStackTrace();
                }
        }
```

```
}
```

24). What is a RequestDispatcher object?

RequestDispatcher is an object which accepts the request from the client and redirects them to any resource(Servlets, JSP, HTML etc). The servlet container creates the RequestDispatcher object.

25). What is session management?

Session management is a mechanism for maintaining state across multiple HTTP requests. This is managed by the web container. In other words it is a technique to hold some values passed by user across multiple HTTP requests arising out from a single browser instance.

26). How long session life cycle is managed?

Session life cycle is managed for each web browser instance opened and it exists till the browser is closed or till the session time outs(set in the server configurations).

27). What is the need for session?

HTTP is a stateless protocol. So if developers need to develop pages where needs to maintain the application user's state across multiple requests he can use session. For example, he can store the following information: Login name, users state/city, user ID number etc.

28). How is session tracked for a user?

Each user session is tracked by unique ID called JSESSIONID. This is similar to how a employee is identified in a organization using employee id. JSESSIONID will be generated the first time the user visits a site. JSESSIONID will be generated for each browser instance.

29).What is a cookie?

It is a simple piece of textual information(in key value pair format) stored on the client(browser machine). Cookies information are returned to the server with every request from the client.

30). How cookies are identified in client machine?

The browser matches the cookies present with the site URL. If a match is found, that cookie is returned with the request.

31). What are the steps for using cookie?

Step i: Create the cookie object.

Step ii: Set the cookie object to the HTTP response.

Step iii: Read the cookies from the next HTTP request.

Step iv: Validate the cookie value for session tracking.

32). How cookie is created?

Cookies are created using the Cookie class of Servlet API.

Cookie cookie = new Cookie(identifier,value);

Where identifier is the name of the state information, value represents the value of the state.

Example: Cookie cookie = new Cookie("username","ABC");

33). How cookies are set to the response?

Since cookies are stored at the client, cookies are set to the response object and sent to the client using the addCookie() method of the HTTPServletResponse interface.

Cookie cookie = new Cookie("username","ABC");

response.addCookie(cookie);

34). How cookie values are read from request?

Cookies stored in the client will be sent to the server along with the HTTP request, each time the client requests a page. The cookies in request object can be read using the getCookies() of the HTTPServletRequest interface.

Cookie cookie[] = request.getCookies();

35).What are the advantages of using cookie?

-> It is easy to develop and maintain.

-> Less overhead to the server since data is stored in the client.

-> It minimizes the server memory usage.

36). What are the disadvantages of using cookie?

Size and number of cookies stored are limited.

Stored as plain text in a specific directory, everyone can view and modify them. So it is not secured.

It is browser dependent, so if client has disabled cookies this can lead to erroneous behaviour of the application.

37). What is a session object?

Session object is a container used object for storing user states in server.

The session object lifecycle is maintained by web container.

The Servlet API HTTPSession interface provides features for session tracking.

HTTPSession objects are objects used for storing client session information.

38). How are session values stored?

Session values are stored in key/value format. Each value would have a name bound to it for retrieving the values.

39How session object is accessed?

Session object associated with a user session is read from the user HTTP request using the getSession() method of the HTTPServletRequest interface.

HTTPSession session = request.getSession();

40).How to set values in HTTP Session?

We can set values in session object using setAttribute() method of the HTTPSession interface.

HTTPSession hs = request.getSession();

hs.setAttribute(String attributeName, Object value);

Let's take a web mail application where user Name and location are stored.

HTTPSession session = request.getSession();

session.setAttribute("user","ABC");

session.setAttribute("Location","Bangalore");

41). How to retrieve attributes from session?

We can retrieve attributes from the session object using getAttribute() method of the HTTPSession interface.

HttpSession hs = request.getSession();

Object var = hs.getAttribute(String attributeName);

Taking a web mail application where user Name and location are stored and retrieving the values.

HttpSession hs = request.getSession();

String userName = (String)hs.getAttribute("User");

String location = (String)hs.getAttribute("Location");

42). How to remove attributes from session?

Values can be removed from the session object using removeAttribute() method of the HTTPSession interface.

HTTPSession session = request.getSession();

session.removeAttribute(String attributeName);

Example: Taking a web mail application where user Name and location are stored. Assume we need to remove the userName attribute "Tim" from the session, the following API needs to be fired on HTTP request object.

HttpSession hs = request.getSession();

session.removeAttribute("User");

43). How to invalidate a session?

Session invalidating is the process of unbinding the session object from the user thereby removing all the previously stored data in the session and freeing the memory. To invalidate a session we call the invalidate() method.

44). When session invalidate() method is used?

This is typically used when user logs off from a web application to free up the memory utilized by the session object.

HttpSession hs = request.getSession();

hs.invalidate();

45). Why to avoid storing bulky objects in session?

Session objects are stored in server and it utilizes memory. So store only the needed information in sessions. Avoid bulky objects in session this can result in "out of memory error" resulting in a application crash.

46). What are the advantages of using HTTP Session?

Simple to use since it is managed in the server. It can function even when the cookie functionality is turned off in the browser.

47). What is the difference between HTTP Get and HTTP Post?

HTTP Get:

1. User entered information is appended in a URL string.

2. It can send only a limited amount of data.

3. The browser caches the data.

HTTP Post:

1. User entered information is send as data as a part of the message body not appended to URL.

2. It can send any amount of data.

3. The data's are not cached.

48). What are the tasks performed by web container when a request comes for a servlet?

Tasks performed by web container are:

1)It loads the servlet class.

2) It instantiates the servlet.

3) It initializes the servlet instance.

4) It passes request to the servlet instance.

5) Finally it sends response to the client.

49).What is a web server?

A computer program responsible for accepting HTTP requests and serving them HTTP responses along with data, which usually are web pages such as HTML pages. They are repository of web pages.

50). What are the features of web server?

Features of web server:

Web server program works by accepting HTTP requests from the client, and providing an HTTP response to the client.

It is a host for all server side scripting like servlet,JSP,ASP,CGI etc.

51). What is an application server?

An application server is a software engine that delivers applications to clients. Moreover, an application server handles most of the business logic and data access of the application.

52). What are the features of application server?

Features of application server:

Application servers enable applications to intercommunicate with dependent applications, like web servers, database management systems and chart programs.

Portals are common application server mechanism by which a single point of entry is provided to multiple devices.

EJB's are hosted in an application server.

53). How to use ServletConfig?

ServletConfig is passed as an argument to the init() method of the servlet by the web container during servlet initialization. Hence to use ServletConfig we need to override the method:

public void init(ServletConfig config) throws ServletException

54). How ServletConfig object is loaded and initialized?

Step i: When a request to a servlet comes for the first time from client or when the application is deployed the web container will initialize the servlet.

Step ii: The web container will parse the web.xml file and read the configurations of the servlet to be initialized.

Step iii: The web container then will create a ServletConfig object and load the configuration details of the servlet in the object.

55). What is the use of HTTPServletRequest Interface?

This interface contains methods to access request information by the HTTP servlets. It is used by servlets to access the parameters sent by client as part of HTTP request from the browser. It is used to access client information like port number, client protocol and form field values.

56). What is the use of HTTPServletResponse interface?

This interface contains methods for managing the response send by an HTTP servlet to the client. It also contains a set of status codes for sending the request status to the client.

57) List the do-XXXX methods in HTTPServlet?

void doGet(HTTPServletRequest req, HTTPServletResponse res)

void doPost(HTTPServletRequest req, HTTPServletResponse res)

void doHead(HTTPServletRequest req, HTTPServletResponse res)

void doOptions(HTTPServletRequest req, HTTPServletResponse res)

void doPut(HTTPServletRequest req, HTTPServletResponse res)

void doTrace(HTTPServletRequest req, HTTPServletResponse res)

void doDelete(HTTPServletRequest req, HTTPServletResponse res)

58) List different types of HTTP requests?

get, post, head, options, put, trace, delete.

59) What are the two packages that deal with servlets?

Servlet API exists in two packages:

javax.servlet: The classes and interfaces in javax.servlet are not protocol dependent. e.g. It can support different protocols like HTTP, FTP.

javax.servlet.http: The classes and interface in this package are specific for requests using HTTP protocol. Some of the classes and interfaces in this package extend those of javax.servlet package.

60).What are the features of servlets?

Security: A Web container provides a runtime environment for executing a servlet. Servlets inherit the security feature provided by the web container. This allows developers to focus on the servlet functionality and leave the security issues to the Web container to handle.

Session management: It is the mechanism of tracking the state of a user across multiple requests. A session maintains the client identity and state across multiple requests.

Instance persistence: Servlets help to enhance the performance of the server by preventing frequent disk access. For example, if a customer logs on to an online banking site, the customer can perform activities, such as checking for the balance or applying for a loan. The account number of the customer will be validated at every stage from the database. Instead of every time checking the account number against the database, servlets retain the account number in the memory till the user logs out of the Web site.

It is platform and server independent.

61) Explain the web application directory structure?

web application's directory structure has document root which contains JSP files, HTML files, JavaScript files and static files such as image files. There is a directory WEB-INF under the document root which contains configuration files related to application. This configuration files can't be served directly to the client. The WEB-INF directory contains:

/WEB-INF/classes/* : It contains compiled class files

/WEB-INF/lib/*.jar : It contains different library jar files

/WEB-INF/web.xml : It is a deployment descriptor that specify the web application configuration.

62). How Send Redirect works?

User requests a URL to the server.

Server sends the browser a status to redirect to a new URL.

Browser requests a new URL and sends a new request.

63). What are session management techniques?

The following are the session management techniques. These are used for managing the data's between pages of the user in a session.

Hidden Form fields

URL Rewriting

Cookies

Session object

64).What is a Hidden Field?

Hidden Fields are nothing but normal HTML form element with type Hidden to store the user data across HTTP request.

65).How Hidden Form field is used for session tracking?

The user state to be preserved across HTTP request can be stored as a hidden elements in the HTML pages which can be read from other pages.

66). How to access Hidden form field values?

Hidden form field values can be read using request.getParameter("key") method.

67). What are the advantages of Hidden form field?

It does not consume any memory space in web server as it is stored in the HTML pages.

It works even if users disable cookies.

68). What are the disadvantages of Hidden form field?

It can be used only with HTML forms.

This is not secured.

Very complex to develop and maintain as all the pages which needs state information needs to be implemented for the hidden fields.

69). What is URL rewriting?

The mechanism by which the user state or information's are appended to the URL for tracking the state/session of the user. Example: The userName is appended to the URL.

http://ww.a4academics.com/SuccessServlet?userName="Tim"&location="India"

where, userName and location are user information set as parameter in the URL.

70). What are the advantages of URL rewriting?

Every data is appended to the URL. So it is easy to debug.

It works even if users disable cookies.

71). What are the disadvantages of URL rewriting?

URL length is a limitation, so we cannot store information beyond a limit.

The URL contains data so it is not secured.

Difficult to maintain in large application since in each page the URL should be rewritten to carry the data.

72). What is a servlet filter?

Servlet filter are programs that runs on the server which intercepts HTTP request/response for transformations/processing/filtering.

73). How does filter work?

Steps of operation of a servlet filter:

Step 1: Intercepts the HTTP request and pre-process the request.

Step 2: Invokes the requested resource with the pre-processed HTTP request.

Step 3: Intercepts the HTTP response and transforms it.

Step 4: The transformed HTTP response will be sent back to the client.

74). What are the usage of filters?

Used for authenticating all HTTP request coming to the server. Example: All requests to the application will be authenticated and authorized against stored user credentials.

All the HTTP requests can be logged and audited in a flat file or database for tracking users of a web application. Example: The user credentials can be logged as "Ron logged in at 8:00 PM and he transferred 10,000 dollars from the account.

To perform some formatting of HTTP response, say display a message in the header (or) bread crumbs for all the pages of the application. Example: All the HTML pages displayed in the client should have the navigation breadcrumb like "Home Page> Savings Account> Account Summary"

75). What are the steps to create a filter?

Step 1: Create a class that implements the filter interface.

Step 2: Implement the doFilter method with the pre/post processing logic.

Step 3: Invoke the doFilter method of the FilterChain object.

Step 4: Register the filter with the appropriate servlets.

76).What is doFilter method?

public void doFilter(ServletRequest request, ServletResponse response,FilterChain chain) throws ServletException, IOException

This method is executed every time when a servlet (or) JSP associated with this filter is invoked. This contains the filtering or pre and post processing logic.

77). Why do we override doFilter method?

It is overridden to perform one or more of the following tasks:

Transform the incoming request.

Transform the response before being sent to the client.

It invokes the doFilter method of the FilterChain interface.

78).What is filter chaining?

Filter chaining is the process of applying more than one filter to a servlet.

79). How servlet filter chain works?

Multiple filters can be chained.the Order is dictated by the order of <filter> elements in the deployment descriptor.

The first filter of the filter chain is invoked by the container via doFilter(ServletRequest req, ServletResponse res, FilterChain chain) method. The filter then perform whatever filter logic and then call the next filter in the chain by calling chain.doFilter(...) method.

The last filter's call to chain.doFilter() which ends up calling service() method of the Servlet.

80). How to send error message to client?

Error message can be send to client by using methods:

public void sendError(int status_code)

or

public void sendError(int status_code, String message)

Status_code describes the type of error occurred, message describes the error and sends the error response to the client.

81). Explain HTTP request briefly?

An HTTP request has three parts:

A request line

One or more request headers

A message

A request line looks like: GET /WelcomeProject/Home.html HTTP/1.1

1st token is the name of the HTTP method, which is GET in this case.

2nd token is the URI, that gives information about the location of the resource to be gotten.

3rd token is the version of HTTP to be used.

82). Explain HTTP response briefly?

An HTTP response has three parts:

A response line

One or more response headers

A message

A response line looks like: HTTP/1.1 200 OK

1st token is the HTTP version.

2nd token is one of the many predefined status codes.

3rd token is an English description of the status code.

83) Explain MVC design pattern?

Model-View-Controller (MVC) is a design pattern which divides a software application into three parts namely model, view and controller.

A model deals with behaviour of the application. It contains the data and business logic of the application. It notifies views and controllers when there is change in its state.

A view renders the information to the user so that it looks attractive and appealing. It takes information from the model using which it generates output.

A controller takes input from user and sends command to model or view. It controls the flow of application.

84). How MVC works?

i): The controller servlet handles the user's request.

ii): The controller servlet then invokes the model and performs the business logic or data retrieval.

iii): The model returns back the data (response) back to the controller.

iv): The controller then transfers the data to the view.

v): The view renders the data in the appropriate format requested by the client.

85).What are the benefits of MVC?

Clear separation of layers, modular and easy to maintain. Example: The model holds business logic or data manipulation logic and view holds the presentation logic. System is modular and easy to maintain as developer know which logic is present in which layer.

Supports parallel development so reduces development time. Example: The view and model can be developed by different teams and finally integrated. This reduces development time to market.

Flexible for future enhancements. Example: Any changes done like database up gradation or change in business logic it is enough if the model is changed without changing the view or controller.

```
while(!(succeed = try()));
```