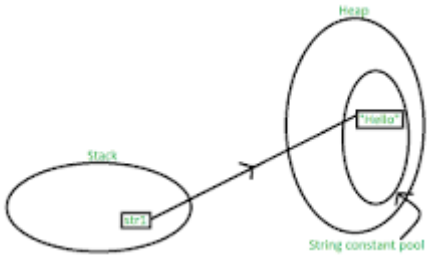
1)What is String constant pool?Why string is immutable?difference between String,StringBuffer,StringBuilder?Difference between == operator and .equals() method?Explain the difference between jdk,jre,jvm?why java is platform independent?

String constant pool:



A string constant pool is a separate place in the heap memory where the values of all the strings which are defined in the program are stored. When we declare a string, an object of type String is created in the stack, while an instance with the value of the string is created in the heap.

String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

Parameter	String	StringBuilder	StringBuffer
mutability	immutable	mutable	mutable
Storage	String constant pool	heap	heap
Thread safety	Not used in the threaded environment as it is immutable	Not thread-safe so it is used in a single-threaded environment	Thread-safe so it is used in a multi-threaded environment
Speed	Comparably slowest	Comparably fastest	Faster than String but Slower than StringBuilder

== operator	Equals() Method
== is considered an operator in Java.	Equals() is considered as a method in Java.
It is majorly used to compare the reference values and objects.	It is used to compare the actual content of the object.

We can use the == operator with objects and primitives.	We cannot use the equals method with primitives.
The == operator can't compare conflicting objects, so at that time the compiler surrenders the compile-time error.	The equals() method can compare conflicting objects utilizing the equals() method and returns "false".
== operator cannot be overridden.	Equals() method and can be overridden.

JDK&JRE&JVM

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.

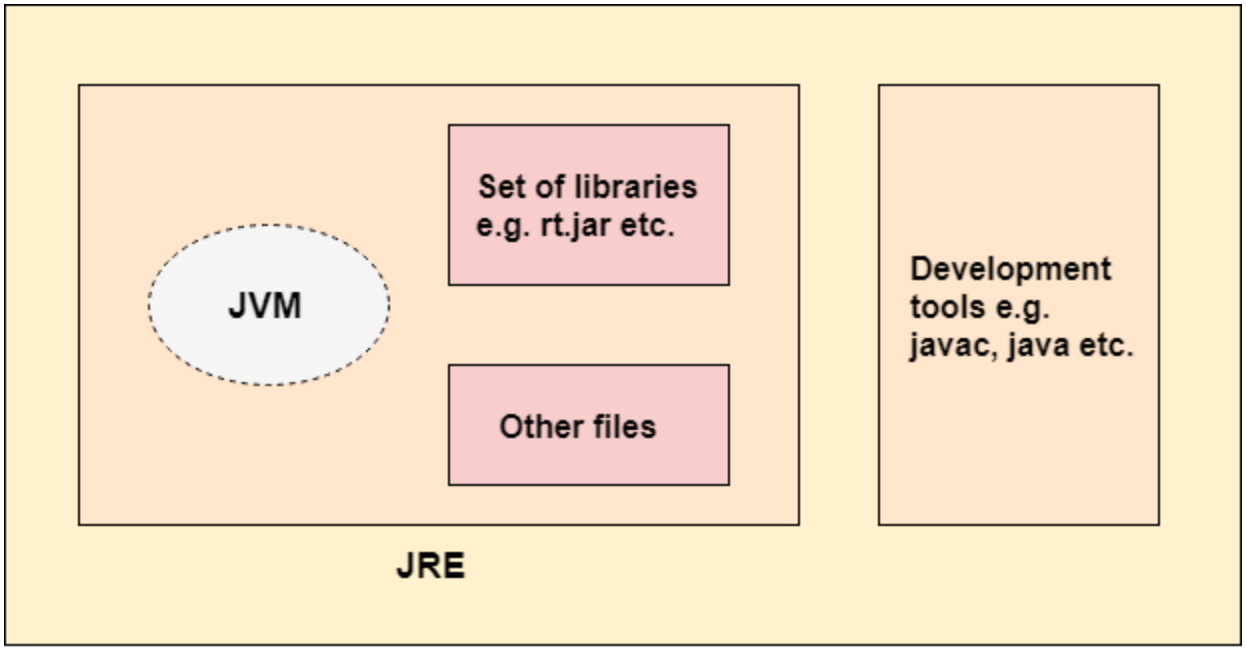
JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

- Standard Edition Java Platform
- Enterprise Edition Java Platform
- Micro Edition Java Platform

The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.



Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides a software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).

2)Difference between Class and Object?Explain in detail OOP's Concept?Explain static and dynamic polymorphism with examples?Difference between static and instance method with examples?Difference between this and super?Difference between method and constructor?Difference interface and Abstract class?What is the significance of public,private,protected classes?

Difference between Class and Object:

Class	Object
1) Class is a collection of similar objects	1) Object is an instance of a class
2) Class is conceptual (is a template)	2) Object is real
3) No memory is allocated for a class.	3) Each object has its own memory
4) Class can exist without any objects	4) Objects can't exist without a class
5) Class does not have any values associated with the fields	5) Every object has its own values associated with the fields

---

Java - What is OOPS?

OOPS stands for Object-Oriented Programming System.

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Object

- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class

Collection of objects is called **class**. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Example:

```
class Student{
 //defining fields
 int id;//field or data member or instance variable
 String name;  //creating main method inside the Student class
 public static void main(String args[]){
  //Creating an object or instance
```

```
 Student s1=new Student();//creating an object of Student
 //Printing values of the object
 System.out.println(s1.id);//accessing member through reference variable
 System.out.println(s1.name);
 }
}
```

Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Program:

```
class Employee
{
    float salary=40000;
}
class Programmer extends Employee
{
     int bonus=10000;
     public static void main(String args[])
    {
        Programmer p=new Programmer();
        System.out.println("Programmer salary is:"+p.salary);
        System.out.println("Bonus of Programmer is:"+p.bonus);
    }
```

}

Polymorphism

If *one task is performed in different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

static polymorphism:

This is achieved through method overloading. Method overloading means there are several methods present in a class having the same name but different types/order/number of parameters.

At compile time, Java knows which method to invoke by checking the method signatures. So, this is called **compile time polymorphism** or **static binding**. The concept will be clear from the following example:

```
public class StaticBindingTest {


   public static void main(String args[])  {
      Collection c = new HashSet();
      StaticBindingTest et = new StaticBindingTest();
      et.sort(c);


   }
```

```
    //overloaded method takes Collection argument
    public Collection sort(Collection c){
        System.out.println("Inside Collection sort method");
        return c;
    }
   //another overloaded method which takes HashSet argument which is subclass
    public Collection sort(HashSet hs){
        System.out.println("Inside HashSet sort method");
        return hs;
    }
}
```

Dynamic Polymorphism:

Suppose a subclass overrides a particular method of the superclass. The polymorphic nature of Java will use the overriding method. Let's say we create an object of the subclass and assign it to the superclass reference. Now, if we call the overridden method on the superclass reference then the subclass version of the method will be called.

Have a look at the following example:

```
public class X
{
    public void methodA() // Base class method
    {
        System.out.println ("hello, I'm methodA of class X");
    }
```

```
}

public class Y extends X
{
    public void methodA() // Derived Class method
    {
        System.out.println ("hello, I'm methodA of class Y");
    }
}
public class Z
{
public static void main (String args []) {

    //this takes input from the user during runtime
    System.out.println("Enter x or y");
    Scanner scanner = new Scanner(System.in);
    String value= scanner.nextLine();

    X obj1 = null;
    if(value.equals("x"))
        obj1 = new X(); // Reference and object X
    else if(value.equals("y"))
        obj2 = new Y(); // X reference but Y object
```

```
    else
        System.out.println("Invalid param value");

    obj1.methodA();
}
}
```

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone calls, we don't know the internal processing.

Program:

```
abstract class Bike
{
        abstract void run();
}
class Honda4 extends Bike
{
        void run()
        {
                System.out.println("running safely");
        }
        public static void main(String args[])
        {
                Bike obj = new Honda4();
                obj.run();
```

```
    }
}
```

Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

A java class is an example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

program:

```
    //A Account class which is a fully encapsulated class.
    //It has a private data member and getter and setter methods.
class Account
 {
    //private data members
    private long acc_no;
    private String name,email;
    private float amount;
    //public getter and setter methods
    public long getAcc_no()
     {
       return acc_no;
    }
    public void setAcc_no(long acc_no)
    {
```

```
 this.acc_no = acc_no;
}
public String getName()
{
        return name;
}
public void setName(String name)
 {
         this.name = name;
}
public String getEmail()
 {
         return email;
}
public void setEmail(String email)
{
        this.email = email;
}
public float getAmount()
{
   return amount;
}
public void setAmount(float amount)
{
         this.amount = amount;
```

```
    }

}
```

Difference Between this and super keyword:

The following table describes the key difference between this and super:

THIS	SUPER
1.The current instance of the class is represented by this keyword.	The current instance of the parent class is represented by the super keyword.
2.In order to call the default constructor of the current class, we can use this keyword.	In order to call the default constructor of the parent class, we can use the super keyword.
3.It can be referred to from a static context. It means it can be invoked from the static context.	It can't be referred to from a static context. It means it cannot be invoked from a static context.
4.We can use it to access only the current class data members and member functions.	We can use it to access the data members and member functions of the parent class.

Difference between method and constructor:

Method	Constructor
1.Name of constructor must be same with of the Class	But there are no such requirements for methods in java.
2.It doesn't have any return type.	It has a return type and returns something unless it's void.
3.Constructors are chained and they are called in a particular order.	There is no such facility for methods.

Difference between Abstract class and Interface :

Abstract class	Interface
1.Abstract classes are fast	Interfaces are slow
2.Abstract class can extends only one class	Interfaces can implements several interfaces
3.You can define fields as well as constants	You cannot define fields in an interface
4.A single abstract class can extend one & only one interface.	A single interface can extend multiple interfaces.

Significance of public,private,protected classes:

public: is a Java keyword which declares a member's access as public. Public members are visible to all other classes. This means that any other class can access a public field or method. Further, other classes can modify public fields unless the field is declared as final .

Private: The private access modifier is specified using the keyword private. The methods or data members declared as private are accessible only within the class in which they are declared. Any other class of the same package will not be able to access these members.

Protected: Variables, methods and constructors, which are declared protected in a superclass can be accessed only by the subclasses in another package or any class within the package of the protected members' class. The protected access modifier cannot be applied to class and interfaces.

3)Difference between Checked and unchecked exceptions?Give 1 example on each?Difference between final,finally,finalize()?Give 1 example on each?Difference between throw and throws?Give 1 example on each?Write a code to show NullPointerException?

CHECKED EXCEPTIONS	UNCHECKED EXCEPTIONS
1.Exceptions that are checked and handled at compile time are checked exceptions.	Exceptions that are not checked and handled at compile time are unchecked exceptions.

2.They are direct subclasses of Exceptions but do not inherit from runtime Exceptions.	They are a direct subclass of the runtime exceptions class.
3.The program gives a compilation error if a method throws a checked exception and the compiler is not able to handle the exception on its own.	The program compiles fine because the exception escapes the notice of the compiler.Exceptions occur due to errors in programming logic.
4.A checked Exception occurs when the chances of failure are too high.	Unchecked Exceptions occur mostly due to programming mistakes.
5.Common checked exceptions include IOexceptions,DateAccessExceptions,IllegalAccessExceptions,InterruptedExceptions,etc;	Common unchecked exceptions include ArthematicExceptions,InvalidClassExceptions,NullPointerException,etc;

final,finally,finalize():

Final:final is the keyword and access modifier which is used to apply restrictions on a class, method or variable.

Example:

```
public class FinalDemo
{
   final int age = 18;//declaring final variable
   void display() {
   age = 55; // reassigning value to age variable gives compile time error
   }

   public static void main(String[] args)
  {
   FinalDemo obj = new FinalDemo();
   obj.display();
   }
}
```

Finally:finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not.

Example:

```
public class FinallyDemo
{

public static void main (String args[])
 {
```

```
try
   {

System.out.println ("Inside try block");

          // below code throws divide by zero exception
     int data = 25 / 0;

System.out.println (data);

}
     // handles the Arithmetic Exception / Divide by zero exception
     catch (ArithmeticException e)
   {

System.out.println ("Exception handled");

System.out.println (e);

}
     // executes regardless of exception occurred or not
     finally
   {

System.out.println ("finally block is always executed");

}
System.out.println ("rest of the code...");

}
}
```

Finalize:finalize is the method in Java which is used to perform clean up processing just before an object is garbage collected.

Example:

```
public class FinalizeExample {

    public static void main(String[] args)

   {

      FinalizeExample obj = new FinalizeExample();
```

```
        // printing the hashcode
        System.out.println("Hashcode is: " + obj.hashCode());
        obj = null;
        // calling the garbage collector using gc()
        System.gc();
        System.out.println("End of the garbage collection");
    }
   // defining the finalize method
    protected void finalize()
    {
        System.out.println("Called the finalize() method");
    }
}
```

Difference between throw and throws:

Throw	Throws
1.Java throw keyword is used to throw an exception explicitly in the code, inside the function or the block of code.	Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code.
2.Type of exception Using throw keyword, we can only propagate unchecked exceptions i.e., the checked exception cannot be propagated using throw only.	Using throws keyword, we can declare both checked and unchecked exceptions. However, the throws keyword can be used to propagate checked exceptions only.
3.The throw keyword is followed by an instance of Exception to be thrown.	The throws keyword is followed by class names of Exceptions to be thrown.
4.Throw is used within the method.	Throws are used with the method signature.
5.We are allowed to throw only one	We can declare multiple exceptions using

exception at a time i.e. we cannot throw multiple exceptions.	throws keywords that can be thrown by the method. For example, main() throws IOException, SQLException.

Throw example:

```
public class TestThrow {

   //defining a method

   public static void checkNum(int num) {

      if (num < 1) {

         throw new ArithmeticException("\nNumber is negative, cannot calculate square");

      }

      else {

         System.out.println("Square of " + num + " is " + (num*num));

      }

   }

   //main method

   public static void main(String[] args) {

         TestThrow obj = new TestThrow();

         obj.checkNum(-3);

         System.out.println("Rest of the code..");

   }

}
```

Throws example:

```
public class TestThrows {

   //defining a method

   public static int divideNum(int m, int n) throws ArithmeticException {

      int div = m / n;
```

```
        return div;
    }
    //main method
    public static void main(String[] args) {
        TestThrows obj = new TestThrows();
        try {
            System.out.println(obj.divideNum(45, 0));
        }
        catch (ArithmeticException e){
            System.out.println("\nNumber cannot be divided by 0");
        }

        System.out.println("Rest of the code..");
    }
}
```

Null pointer Exception Example:

```
class Main
{
        public static void main(String []args)
        {
            String s=null;
            System.out.println(s.length());
        }
}
```

4)How many ways we can create a thread?best approach what?Difference between sleep and wait methods?Explain different methods present in Thread class?How to Create a Deadlock Scenario Programmatically?Explain the life cycle of a

thread in Java?Explain synchronization?Difference between static and synchronized block?

Thread: A thread in Java is a lightweight process or the smallest unit of a process. Each thread performs a different task and hence a single process can have multiple threads to perform multiple tasks. Threads use shared resources and hence requires less memory and less resource usage. Threads are an important concept in multithreading. The main method itself is a thread that runs the Java program.

There are two ways to create a thread:There are two ways to create a thread:There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

Best approach to define a Thread:

- Among the 2 ways of defining a Thread, implementing Runnable approach is always recommended.
- In the 1st approach our class should always extend Thread class there is no chance of extending any other class hence we are missing the benefits of inheritance.
- But in the 2nd approach while implementing Runnable interface we can extend some other classes also. Hence implementing a Runnable mechanism is recommended.

Difference between sleep and wait methods:

Wait()	Sleep()
1.Wait() method belongs to the Object class.	Sleep() method belongs to the Thread class.
2.Wait() method releases lock during Synchronization.	Sleep() method does not release the lock on the object during Synchronization.

3.Wait() should be called only from the Synchronized context.	**There is no need to call sleep() from a Synchronized context.**
4.Wait() is not a static method.	**Sleep() is a static method.**
5.Wait() Has Three Overloaded Methods: ● **wait()** ● **wait(long timeout)** ● **wait(long timeout, int nanos)**	**Sleep() Has Two Overloaded Methods:** ● **sleep(long millis)millis: milliseconds** ● **sleep(long millis,int nanos) nanos: Nanoseconds**
6.public final void wait(long timeout)	**public static void sleep(long millis) throws Interrupted_Execption**

Thread class contains the following methods:

- **yield():**yield() method causes "to pause current executing Thread for giving the chance of remaining waiting Threads of the same priority".

- **join():**If a Thread wants to wait until completing some other Thread then we should go for
  join() method.

- **sleep():**If a Thread doesn't want to perform any operation for a particular amount of time then we should go for sleep() method.

- **start():**Causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread.

- **toString():**Returns a string representation of this thread, including the thread's name, priority, and thread group

- **run():**If this thread was constructed using a separate Runnable run object, then that Runnable object's run method is called; otherwise, this method does nothing and returns.

- **checkAccess():**Determines if the currently running thread has permission to modify this thread.

- **clone():**Throws CloneNotSupportedException as a Thread can not be meaningfully clone.Throws CloneNotSupportedException as a Thread can not be meaningfully cloned.

★ Deadlock Scenario Programmatically

```
public class TestDeadlockExample1
{
   public static void main(String[] args)
   {
      final String resource1 = "ratan jaiswal";
      final String resource2 = "vimal jaiswal";
      Thread t1 = new Thread() // t1 tries to lock resource1 then
resource2
      {
         public void run() {
         synchronized (resource1)
         {
            System.out.println("Thread 1: locked resource 1");
            try { Thread.sleep(100);} catch (Exception e) {}
            synchronized (resource2)
            {
               System.out.println("Thread 1: locked resource 2");
            }
         }
      }
   };
   Thread t2 = new Thread() // t2 tries to lock resource2 then
resource1
   {
      public void run()
      {
         synchronized (resource2)
         {
            System.out.println("Thread 2: locked resource 2");

            try { Thread.sleep(100);} catch (Exception e) {}

            synchronized (resource1)
            {
               System.out.println("Thread 2: locked resource 1");
```

```
            }
        }
    }
};
t1.start();
t2.start();
}
}
```

★ life cycle of a thread in Java:

A thread goes through various stages in its lifecycle. For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread.

Following are the stages of the life cycle –

- **New** – A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
- **Runnable** – After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting** – Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. Thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed Waiting** – A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the

runnable state when that time interval expires or when the event it is waiting for occurs.

- **Terminated (Dead)** – A runnable thread enters the terminated state when it completes its task or otherwise terminates

synchronization:

Synchronization is a process of handling resource accessibility by multiple thread requests. The main purpose of synchronization is to avoid thread interference. At times when more than one thread tries to access a shared resource, we need to ensure that resource will be used by only one thread at a time.

General Syntax:

```
synchronized(object)

{

//statement to be synchronized

}
```

Difference between static and synchronized block:

static block:

- **A static method can be invoked without the need for creating an instance of a class.**
- **Static method can access a static data member and can change the value of it.**
- **Static block is used to initialize the static data member.**
- **It is executed before the main method at the time of classloading.**

- **Syntax:**

```
static
{
        //statement that executes before the main method.
}
```

synchronized block:

- Synchronized block can be used to perform synchronization on any specific resource of the method.
- Suppose we have 50 lines of code in our method, but we want to synchronize only 5 lines, in such cases, we can use a synchronized block.
- If we put all the codes of the method in the synchronized block, it will work the same as the synchronized method.

Syntax:

```
synchronized (object reference expression)

{

        //code block

}
```

5)Internal working of HashMap?What is ConcurrentHashMap giving one example program?when we go for list ,set,map?Frequent operation is searching what we need to use?

What is HashMap:

HashMap is a part of the Java collection framework. It uses a technique called Hashing. It implements the map interface. It stores the data in the pair of Key and Value. HashMap contains an array of the nodes, and the node is represented as a class. It uses an array and LinkedList data structure internally for storing Key and Value. There are four fields in HashMap.

We must be aware of hashCode() and equals() methods before understanding the internal process of Hashmap.

equals(): It checks the equality of two objects. It compares the Key, whether they are equal or not. It is a method of the Object class. It can be overridden. If you override the equals() method, then it is mandatory to override the hashCode() method.

- **hashCode():** This is the method of the object class. It returns the memory reference of the object in integer form. The value received from the method is used as the bucket number. The bucket number is the address of the element inside the map. Hash code of null Key is 0.
- **Buckets:** Arrays of the node are called buckets. Each node has a data structure like a LinkedList. More than one node can share the same bucket. It may be different in capacity.

Insert Key, Value pair in HashMap

We use the put() method to insert the Key and Value pair in the HashMap. The default size of HashMap is 16 (0 to 15).

Example

In the following example, we want to insert three (Key, Value) pairs in the HashMap.

```
HashMap<String, Integer> map = new HashMap<>();

map.put("Aman", 19);

map.put("Sunny", 29);

map.put("Ritesh", 39);
```

Let's see at which index the Key, value pair will be saved into HashMap. When we call the put() method, then it calculates the hash code of the Key "Aman." Suppose the hash code of "Aman" is 2657860. To store the Key in memory, we have to calculate the index.

Calculating Index

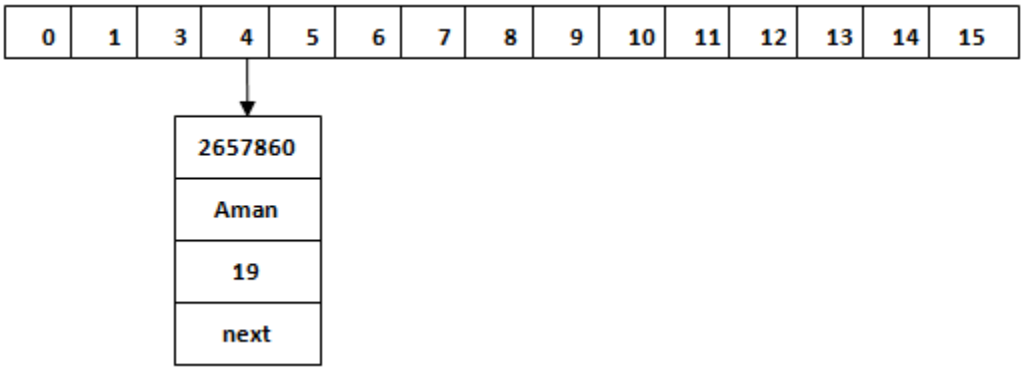
Index minimizes the size of the array. The Formula for calculating the index is:

```
Index = hashcode(Key) & (n-1)
```

Where n is the size of the array. Hence the index value for "Aman" is:

Index = 2657860 & (16-1) = 4

The value 4 is the computed index value where the Key and value will store in HashMap.

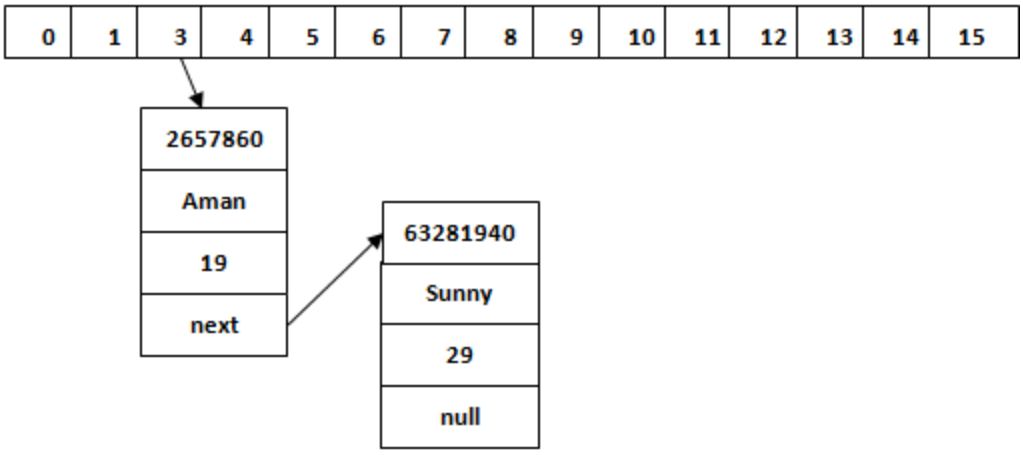


Hash Collision

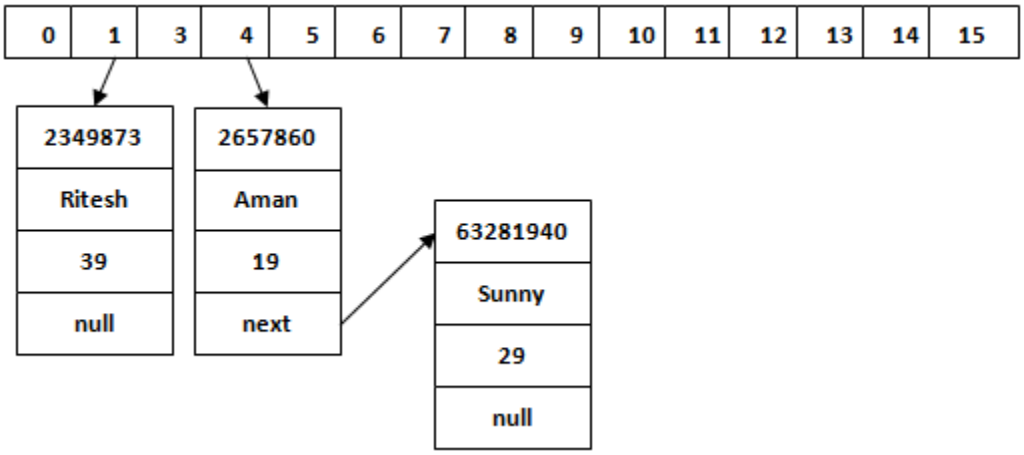
This is the case when the calculated index value is the same for two or more Keys. Let's calculate the hash code for another Key "Sunny." Suppose the hash code for "Sunny" is 63281940. To store the Key in the memory, we have to calculate the index by using the index formula.

Index=63281940 & (16-1) = 4

The value 4 is the computed index value where the Key will be stored in HashMap. In this case, the equals() method checks that both Keys are equal or not. If Keys are the same, replace the value with the current value. Otherwise, connect this node object to the existing node object through the LinkedList. Hence both Keys will be stored at index 4.



Similarly, we will store the Key "Ritesh." Suppose the hash code for the Key is 2349873. The index value will be 1. Hence this Key will be stored at index 1.



get() method in HashMap

get() method is used to get the value by its Key. It will not fetch the value if you don't know the Key. When the get(K Key) method is called, it calculates the hash code of the Key.

Suppose we have to fetch the Key "Aman." The following method will be called.

```
map.get(new Key("Aman"));
```

It generates the hash code as 2657860. Now calculate the index value of 2657860 by using the index formula. The index value will be 4, as we have calculated above. get() method search for the index value 4. It compares the first element Key with the given Key. If both keys are equal, then it returns the value and checks for the next element in the node if it exists. In our scenario, it is found as the first element of the node and returns the value 19.

Let's fetch another Key "Sunny."

The hash code of the Key "Sunny" is 63281940. The calculated index value of 63281940 is 4, as we have calculated for the put() method. Go to index 4 of the array and compare the first element's Key with the given Key. It also compares Keys. In our scenario, the given Key is the second element, and the next of the nodes is null. It compares the second element Key with the specified Key and returns the value 29. It returns null if the next of the nodes is null.

ConcurrentHashMap:

The **ConcurrentHashMap** class introduced in JDK 1.5 belongs to **java.util.concurrent** package, which implements ConcurrentMap as well as to Serializable interface also. ConcurrentHashMap is an enhancement of HashMap as we know that while dealing with Threads in our application HashMap is not a good choice because performance-wise HashMap is not up to the mark.

Key points of ConcurrentHashMap:

- The underlined data structure for ConcurrentHashMap is Hashtable.
- ConcurrentHashMap class is thread-safe i.e. multiple threads can operate on a single object without any complications.
- At a time any number of threads are applicable for a read operation without locking the ConcurrentHashMap object which is not there in HashMap.
- In ConcurrentHashMap, the Object is divided into a number of segments according to the concurrency level.
- The default concurrency-level of ConcurrentHashMap is 16.
- In ConcurrentHashMap, at any time any number of threads can perform retrieval operation but for updates in the object, the thread must lock the particular segment in which the thread wants to operate. This type of locking mechanism is known as **Segment locking or bucket locking**. Hence at a time, 16 update operations can be performed by threads.
- Inserting null objects is not possible in ConcurrentHashMap as a key or value.

Program:

```
import java.util.concurrent.*;

class Main {

        public static void main(String[] args)

        {

        ConcurrentHashMap<Integer,String> m= new
        ConcurrentHashMap<>();//create an instance
        of ConcurrentHashMap

        m.put(100, "Hello"); // Insert mappings using
        put method

    m.put(101, "Geeks");

        m.put(102, "Geeks");

        m.putIfAbsent(101, "Hello");    // Here we cant add
Hello because 101 key is already present in
ConcurrentHashMap object
```

```
        m.remove(101, "Geeks");// We can remove entry because 101 key is associated with For value

        m.putIfAbsent(103, "Hello");    // Now we can add Hello

        m.replace(101, "Hello", "For"); // We can't replace Hello with For

        System.out.println(m);

    }

}
```

When to use List, Set and Map in Java?

- If you do not want to have duplicate values in the database then Set should be your first choice as all of its classes do not allow duplicates.
- If there is a need for frequent search operations based on the index values then List (ArrayList) is a better choice.
- If there is a need of maintaining the insertion order then also the List is a preferred collection interface.
- If the requirement is to have the key & value mappings in the database then Map is your best bet.

Frequent operation is searching what we need to use this:

- **With given restrictions, you should use HashMap. It will give you a quick search, as you wished.**
- **If you care about traversing elements in specific order, you should choose TreeMap (natural order) or LinkedHashMap (insertion order).**
- **If your collection is guaranteed immutable, you can use sorted ArrayList with binary search, it will save you some memory. In this case, you can search only by one specific key, which is undesirable in many real world applications.**
- **Anyway, you should have a really huge number of elements (millions/billions) to feel the difference between O(logN) solutions and O(1) solutions.**

6)Explain java 8 features?

The following are the java 8 features:

★ Lambda expressions
★ Functional interfaces
★ Default methods
★ Predicates
★ Functions
★ Double colon operator(::)
★ Stream API
★ Date and Time API

★ **Lambda expressions:**Lambda Expression is just an anonymous(nameless) function. That means the function which does not have the name,return type and access modifiers.Lambda Expression also known as anonymous functions or closures.

❖ **Program:**

```
@ FunctionalInterface                //It is optional

interface Drawable
```

```
{
  public void draw ();
}

public class LambdaExpressionExample2
{
  public static void main (String[]args)
  {
    int width = 10;

    //with lambda
    Drawable d2 = ()->{
      System.out.println ("Drawing " + width);
    };
    d2.draw ();
  }
}
```

★ **Functional interfaces:** An Interface that contains only one abstract method is known as functional interface. It can have any number of default and static methods. It can also declare methods of object class.Functional interfaces are also known as Single Abstract Method Interfaces (SAM Interfaces).

★ **Program:**

```
@ FunctionalInterface interface sayable

{

  void say (String msg);

}



public class FunctionalInterfaceExample implements sayable

{

  public void say (String msg)

  {

    System.out.println (msg);

  }

  public static void main (String[]args)

  {

    FunctionalInterfaceExample file = new FunctionalInterfaceExample ();

    fie.say ("Hello there");
```

```
    }

  }
```

★ **Default methods:** Java provides a facility to create default methods inside the interface. Methods which are defined inside the interface and tagged with default keywords are known as default methods. These methods are non-abstract methods and can have method bodies.

★ **Program:**

```
interface Sayable

{

 default void say ()  // Default method

 {

  System.out.println ("Hello, this is default method");

 }

 // Abstract method

 void sayMore (String msg);

}



public class DefaultMethods implements Sayable

{

 public void sayMore (String msg)
```

```
{                                   // implementing abstract method
  System.out.println (msg);
}
public static void main (String[]args)
{
  DefaultMethods dmv= new DefaultMethods ();
  dm.say ();                        // calling default method
  dm.sayMore ("Work is worship");         // calling abstract method


}
}
```

★ **Predicates:** It is a functional interface which represents a predicate (boolean-valued function) of one argument. It is defined in the java.util.function package and contains test() a functional method.

★ program:

```
import java.util.function.Predicate;
public class PredicateInterfaceExample
{
        public static void main(String[] args)
        {
                Predicate<Integer> pr = a -> (a > );18 // Creating predicate
```

```
                System.out.println(pr.test(10));    // Calling Predicate method
        }
}
```

★ **Functions:**Functions are exactly same as predicates except that functions can return any type of result but

function should(can)return only one value and that value can be any type as per our requirement.

To implement functions oracle people introduced Function interface in 1.8version.

Function interface present in java.util.function package.

Functional interface contains only one method i.e., apply()

interface function(T,R)

{

public R apply(T t);

}

★ **Double colon operator(::):**functionalInterface method can be mapped to our specified method by using :: (double colon)operator. This is called method reference.Our specified method can be either static method or instance method.FunctionalInterface method and our specified method should have the same argument types ,except that the remaining things like returntype,Methodname,modifiers etc are not required to match.

★ **program:**

```
class Test {
```

```
public static void m1() {

for(int i=0; i<=10; i++) {

System.out.println("Child Thread");}}

public static void main(String[] args) {

Runnable r = Test:: m1;

Thread t = new Thread(r);

t.start();

for(int i=0; i<=10; i++) {

System.out.println("Main Thread");

}

}
```

★ **Stream API:** Introduced in Java 8, the Stream API is used to process collections of objects. A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result.

The features of Java stream are –

- A stream is not a data structure, instead it takes input from the Collections, Arrays or I/O channels.
- Streams don't change the original data structure, they only provide the result as per the pipelined methods.

- Each intermediate operation is lazily executed and returns a stream as a result, hence various intermediate operations can be pipelined. Terminal operations mark the end of the stream and return the result.

Example program:

```
//a simple program to demonstrate the use of stream in java

import java.util.*;

import java.util.stream.*;



class Demo

{

public static void main(String args[])

{



    // create a list of integers

    List<Integer> number = Arrays.asList(2,3,4,5);
```

```
// demonstration of map method

List<Integer> square = number.stream().map(x -> x*x).

                                        collect(Collectors.toList());

System.out.println(square);



// create a list of String

List<String> names =

                    Arrays.asList("Reflection","Collection","Stream");



// demonstration of filter method

List<String> result = names.stream().filter(s->s.startsWith("S")).

                                        collect(Collectors.toList());

System.out.println(result);



// demonstration of sorted method
```

```
List<String> show =

                names.stream().sorted().collect(Collectors.toList());

System.out.println(show);



// create a list of integers

List<Integer> numbers = Arrays.asList(2,3,4,5,2);



// collect method returns a set

Set<Integer> squareSet =

        numbers.stream().map(x->x*x).collect(Collectors.toSet());

System.out.println(squareSet);



// demonstration of forEach method

number.stream().map(x->x*x).forEach(y->System.out.println(y));
```

```
        // demonstration of reduce method

        int even =

        number.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);




        System.out.println(even);

}

}
```

★ **Date and Time API:** until java 1.7version the classes present in java.util package to handle Date and Time(like Date,Calendar,TimeZoneetc) are not upto the mark with respect to convenience and performance.To overcome this problem in the 1.8version oracle people introduced Joda-Time API . This API developed by joda.org and available in java in the form of java.time package.

```
        //program to display System Date and time.

        import java.time.*;

        public class DateTime

        {

          public static void main (String[]args)

          {

            LocalDate date = LocalDate.now ();
```

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

LocalTime time = LocalTime.now ();

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

7)Print the nearest prime to given number

Input:10

Output:7,11//10 below nearest prime number 7 and 10 above11

Program:

```
public class NearestPrime
{
  public static boolean isprime(int number)
  {
    for (int i = 2; i < (number / 2); i++)
     {
          if (number % i == 0)
           {
             return false;
           }
     }
    return true;
  }
  public static void main (String[]args)
  {
    java.util.Scanner s1 = new java.util.Scanner (System.in);
    int input = s1.nextInt ();
    for (int i = 1; i <= input; i++)
    {
              int n= input+i;
              if(isprime(n))
              {
                 System.out.println ("The nearest prime number is:" + n);
                 break;
              }
              n= input-i;
              if(isprime(n))
```

```
            {
                System.out.println ("The nearest prime number is:" +n);
                break;
        }
    }
}
}
```

8)How to swap two numbers without using a third variable?

Swapping Algorithm:

- **STEP 1:** START
- **STEP 2:** ENTER x, y
- **STEP 3:** PRINT x, y
- **STEP 4:** x = x + y
- **STEP 5:** y= x - y
- **STEP 6:** x =x - y
- **STEP 7:** PRINT x, y
- **STEP 8:** END

Example program:

```
import java.util.*;

class Swap   {

 public static void main(String a[]){

System.out.println("Enter the value of x and y");

Scanner sc = new Scanner(System.in);

 /*Define variables*/

int x = sc.nextInt();
```

```
        int y = sc.nextInt();

        System.out.println("before swapping numbers: "+x +" "+ y);

        /*Swapping*/

        x = x + y;

         y = x - y;

        x = x - y;

        System.out.println("After swapping: "+x +"  " + y);

        }

}
```

9)Reverse Number and String program

Input: 123

output:321

Input:coachit

Output:ticalc

Program:

```
public class StringReverse
{
  public static void main (String[]args)
  {
    java.util.Scanner stringswap = new java.util.Scanner
(System.in);
    String str = stringswap.next();
    String nstr = "";
    char ch;
```

```
    System.out.println("The Original string:"+str);
    for (int i=0;i<str.length();i++)
    {
            ch = str.charAt (i);
            nstr = ch+nstr;
    }
    System.out.print("The reversed string:"+nstr);
  }
}
```

10)Fibonacci Series program?Find the factorial of an integer?

★ Fibonacci Program:

```
import java.util.*;
public class Main
{
        public static void main (String[]args)
        {
                ArrayList al=new ArrayList();
                Scanner stringswap = new Scanner (System.in);
                int a=0,b=1;
                System.out.print("How many fibonacci numbers do you want:");
                int n=stringswap.nextInt();
                ArrayList AL=new ArrayList();
                for(int i=1;i<=n;i++)
                {
                        int c=a+b;
                        al.add(a);
                        a=b;
                        b=c;
                }
                System.out.print("The first "+n+" fibonacci numbers are:"+al);
        }
}
```

★ Factorial Program:

```
import java.util.*;
public class FactorialDemo
{
  public static void main (String[]args)
```

```
    {
      Scanner fib= new Scanner (System.in);
      System.out.print("Enter the number:");
      int n=fib.nextInt();
      int fact=1;
      for(int i=1;i<=n;i++)
      {
          fact=fact*i;
      }
      System.out.print("The factorial of "+n+" is:"+fact);
    }
  }
```

11)List contains some Numbers?Print only even numbers from the list?

Program:

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Random;
public class Main {
    static Random rand=new Random();
        public static void main(String[] args)
        {
                List<Integer>list= new ArrayList();
                        for(int i=0;i<=5;i++)
                        {
                          int ra=rand.nextInt(100)+1;
                          list.add(ra);
                        }
                Iterator itr= list.iterator();
                while (itr.hasNext())
                {
                        int j=(int)itr.next();
                        if(j%2==0)
                        System.out.println(j);
                }
        }
}
```

12)Sorting an array in Java?

input:1, 2, 3, -1, -2, 4

output:-2,-1,1,2,3,4

do without using special methods and with special methods?Find the second largest number in the array?Check particular elements present in the array or not?

Program without Sorting technique:

```
        import java.util.*;
public class Main
{
  static int a[] = { -1, -2, -3, 1, 2, 4 };
  static int temp;
  public static void main (String[]args)
  {
    // TODO Auto-generated method stub
    for (int i = 0; i < a.length-1; i++)
      {
            if (a[i] > a[i+1])
              {
                  temp = a[i];
                  a[i] = a[i+1];
                  a[i+1] = temp;
                  i=-1;
              }
      }
            System.out.println (Arrays.toString(a));

  }

}
```

Program with Sorting technique:

```
        import java.util.Arrays;
        class Main
        {
                static void bubbleSort (int array[])// perform the bubble sort {
                int size = array.length;

              for (int i = 0; i < size - 1; i++)// loop to access each array element


                for (int j = 0; j < size - i - 1; j++)// loop to compare array elements


                        if (array[j] > array[j + 1])/*compare two adjacent elements change > to < to
        sort in descending order*/
```

```
            {
                    int temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
             }
    }

    public static void main (String args[])
    {

      int[] data = { -2, 45, 0, 11, -9 };

      // call method using class name
      Main.bubbleSort (data);

      System.out.println ("Sorted Array in Ascending Order:");
      System.out.println (Arrays.toString (data));
    }
}
```

Program to find second largest Number:

```
public class SecondLargest {

public static void main(String[] args) {
        // TODO Auto-generated method stub

        int arr[]= {10,45,84,74,84,85};
        int f=arr[0],s=arr[0],t=arr[0];
        for(int i=0;i<arr.length;i++)
        {
                if(arr[i]>f)
                {
                        t=s;
                        s=f;
                        f=arr[i];
                }
                else if(arr[i]>s)
                {
                        t=s;
                        s=arr[i];
                }
        }
        if(f==s)
        {
```

```
                System.out.print(t);
            }
            else
            {
                System.out.print(s);
            }
        }

}
```

Program to find particular elements present in the array or not:

```
            public class Main
{
    public static void main(String[] args)
    {
       java.util.Scanner s1=new java.util.Scanner(System.in);
            int a[]={2,3,5,25,43,57};
            System.out.print("enter the key value:");
            int key=s1.nextInt();
            int i;
            for(i=0;i<a.length-1;i++)
            {
               if(a[i]==key)
               {
                  System.out.print("found");
                  break;
               }
            }
            if(a[i]!=key)
            {
               System.out.print("Notfound");
            }

    }
}
```

13)Check if two arrays contain the same elements or not?

Program to check the value:

```
    import java.util.*;
    public class Main
    {
            public static void main(String[] args)
```

```
        {
                java.util.Scanner s1=new java.util.Scanner(System.in);
                int a[]={2,3,5,25,43,57};
                int b[]={2,3,5,25,43,57};
                Arrays.sort(a);
                Arrays.sort(b);
                boolean returnVal1 = Arrays.equals(a,b);
                System.out.println(returnVal1);

        }
    }
```

14)Check given Number is perfect or not

Input:Input 6(6 is divisible by 1,2,3)1+2+3

Output:true

Program:

```
import java.util.*;
public class Main
{
    public static void main(String[] args)
    {
       java.util.Scanner s1=new java.util.Scanner(System.in);
        int n=s1.nextInt(),sum=0;
        for(int i=1;i<n;i++)
        {
           if(n%i==0)
           {
              sum=sum+i;
           }
        }
        if(sum==n)
        {
           System.out.print("It is a perfect number");
        }
        else{
           System.out.print("It is a not perfect number");
        }
    }
}
```

15)Armstrong Number program

input:153(1*1*1+5*5*5+3*3*3=153)

Output:true

Program:

```
        import java.util.*;
public class Main
{
        public static void main(String[] args)
        {
           java.util.Scanner s1=new java.util.Scanner(System.in);
                int n=s1.nextInt(),c=0,sum=0;
                int temp=n;
      while(n>0)
      {
        n=n/10;
        c++;
      }
      n=temp;
      while(n>0)
      {
         int r=n%10;
         n=n/10;
         sum=sum+(int)(Math.pow(r,c));
      }
                if(temp==sum)
                {
                   System.out.println("armstrong number");
                }
                else{
                   System.out.println("not a armstrong number");
                }
        }
}
```

16)find the largest and smallest element in the array?

Program:

```
        public class Main
        {
                public static void main(String[] args)
                {
                        int arr[]={76,2,4,7,19,24};
                        int min=arr[0],max=arr[0];
                        for(int i=0;i<arr.length;i++)
                        {
```

```
                    if(min>arr[i])
                    {
                        min=arr[i];
                    }
                    if(max<arr[i])
                    {
                        max=arr[i];
                    }
                }
                System.out.println("This is a minimum number: "+min);
                System.out.println("This is a maximum number: "+max);
            }
        }
```

17)find the number of vowels and consonants count in the String

Input:ramesh

Output:vowels:2,consonants:4

Program:

```
public class Vowelcount
{
    public static void main(String[] args) {
    java.util.Scanner s1=new java.util.Scanner(System.in);
    String name=s1.nextLine();
    int c=0,count=0;
    name=name.toLowerCase();
    for(int i=0;i<name.length();i++)
    {

    if((name.charAt(i)=='a')||(name.charAt(i)=='e')||(name.charAt(i)=='i')||(name.charA
    t(i)=='o')||(name.charAt(i)=='u'))
       {
          count++;
       }
       Else{
          c++;
       }
    }
    System.out.println("This are the vowels in the String "+count);
        System.out.print("This are the consonants in the String "+c);
}
}
```

18)word reverse and First letter capital program

input:this is java
output:Siht Si Avaj
Program:

```
        class ReverseEach {

   static java.util.Scanner sc= new java.util.Scanner(System.in);;

        public static void main (String[]args)
   {
          String str = "sanjeev kumar is a good boy";
          String[] strArray = str.split (" ");
          for (int i = 0; i <strArray.length; i++)
          {
                  char[] s1 = strArray[i].toCharArray ();
                   for (int j = s1.length - 1; j >= 0; j--)
                   {
                           System.out.print (s1[j]);
                   }
                   System.out.print (" ");
          }
   }
}
```

19)Remove space in a String
input:this is java
Output:thisisjjava
Program:

```
        public class Main
{
        public static void main(String[] args) {
           java.util.Scanner s1=new java.util.Scanner(System.in);
                  String str=s1.nextLine();
                   str = str.replaceAll("\\s","");
                   System.out.print(str);


        }
}
```

20)List is the collection of person object Person class contains name,age
From the list of person object remove below 18 age person objects
after removing the Print latest List of person objects?
Do with and without stream api?

```
public class Person

{

    String name;

    int age;

    Person(String name, int age)

    {

                this.name=name;

                this.age=age;

    }

    public static void Person(String[] args)

    {

                List <Person>Person = new ArrayList<Person>();

                Person.add(new Person("Rama",05));

                Person.add(new Person("Raja", 15));

                Person.add(new Person("Mounika",26));

                Person.add(new Person("Akshata", 55));

                Person.add(new Person("Sanjay",25));

                Person.add(new Person("Chandu", 02));
```

```
                Person.stream().filter(P -> P.age >18).collect(Collectors.toList()).forEach(Person
                ->System.out.print(Person.name+" "));

    }

}
```

21)Example program on comparable and comparator?In which scenario do we use comparator?

Java Comparable interface

Java Comparable interface is used to order the objects of the user-defined class. This interface is found in the java.lang package and contains only one method named compareTo(Object). It provides a single sorting sequence only, i.e., you can sort the elements on the basis of a single data member only. For example, it may be rollno, name, age or anything else.

Example program:

```
import java.util.*;

public class Main
{
  public static void main (String args[])
  {
    ArrayList < Student > al = new ArrayList < Student > ();
    al.add (new Student (101, "Vijay", 23));
    al.add (new Student (106, "Ajay", 27));
    al.add (new Student (105, "Jai", 21));

    Collections.sort (al);
    for (Student st:al)
      {
            System.out.println (st.rollno + " " + st.name + " " + st.age);
      }
  }
}
class Student implements Comparable < Student >
{
  int rollno;
  String name;
  int age;
  Student (int rollno, String name, int age)
```

```
 {
   this.rollno = rollno;
   this.name = name;
   this.age = age;
 }

 public int compareTo (Student st)
 {
   if (age == st.age)
     return 0;
   else if (age < st.age)
     return 1;
   else
     return -1;
 }
}
```

Java Comparator interface

Java Comparator interface is used to order the objects of a user-defined class.

This interface is found in the java.util package and contains 2 methods compare(Object obj1,Object obj2) and equals(Object element).

It provides multiple sorting sequences, i.e., you can sort the elements on the basis of any data member, for example, rollno, name, age or anything else.

Scenario: In case a different sorting order is required, then, implement a comparator and define its own way of comparing two instances.

Example program:

```
import java.util.*;
public class NameAgeSort
{
 public static void main (String args[])
 {
   ArrayList < Student > al = new ArrayList < Student > ();
   al.add (new Student (101, "Vijay", 23));
   al.add (new Student (106, "Ajay", 27));
   al.add (new Student (105, "Jai", 21));
   //Sorting elements on the basis of name
   Comparator < Student > cm1 = Comparator.comparing (Student::getName);
```

```
    Collections.sort (al, cm1);
    System.out.println ("Sorting by Name");
    for (Student st:al)
      {
          System.out.println (st.rollno + " " + st.name + " " + st.age);
      }
    //Sorting elements on the basis of age
    Comparator < Student > cm2 = Comparator.comparing (Student::getAge);
    Collections.sort (al, cm2);
    System.out.println ("Sorting by Age");
  for (Student st:al)
      {
          System.out.println (st.rollno + " " + st.name + " " + st.age);
      }
  }
}

class Student
{
  int rollno;
  String name;
  int age;
    Student (int rollno, String name, int age)
  {
    this.rollno = rollno;
    this.name = name;
    this.age = age;
  }

  public int getRollno ()
  {
    return rollno;
  }

  public void setRollno (int rollno)
  {
    this.rollno = rollno;
  }

  public String getName ()
  {
    return name;
  }
```

```
  public void setName (String name)
  {
    this.name = name;
  }

  public int getAge ()
  {
    return age;
  }

  public void setAge (int age)
  {
    this.age = age;
  }

}
```