```java
package com.ravi.arraylist;

import java.util.ArrayList;
import java.util.Collections;

public class ArrayListDemo5
{
    public static void main(String[] args)
    {
        ArrayList<String> cities = new ArrayList<>();

        cities.add("Hyderabad");
        cities.add("Delhi");
        cities.add("Banglore");
        cities.add("Chennai");

        System.out.println("Before sorting: " + cities);

        Collections.sort(cities);
        System.out.println("After sorting (Ascending): " + cities);

        Collections.sort(cities,Collections.reverseOrder());
        System.out.println("After sorting (Descending): " + cities);

    }
}
```

Collections class has provided static method reverseOrder(), return type is Comparator.

```java
package com.ravi.arraylist;


//Program on ArrayList that contains null values as well as we can pass
//the element based on the index position
import java.util.ArrayList;
public class ArrayListDemo6
{
    public static void main(String[] args)
    {
        ArrayList<Object> al = new ArrayList<>(); //Generic type
        al.add(12);
        al.add("Ravi");
        al.add(12);
        al.add(3,"Hyderabad");
        al.add(1,"Naresh");
        al.add(null);
        al.add(11);
        System.out.println(al);  //12 Naresh Ravi  12   Hyderabad  null  11
    }
}
```

```java
package com.ravi.arraylist;

import java.util.ArrayList;
import java.util.List;

record Professor(String name, String specialization)
{
}

class Department
{
    private String departmentName;
    private List<Professor> professors;


    public Department(String departmentName)
    {
        this.departmentName = departmentName;
        professors = new ArrayList<Professor>();  //Composition
    }

    public void addProffesor(Professor professor)
    {
        professors.add(professor);
    }

    public String getDepartmentName()
    {
        return this.departmentName;
    }


    public List<Professor> getProfessors()
    {
        return this.professors;
    }
}

public class ArrayListDemo7
{
     public static void main(String[] args)
     {
        Department dept = new Department("Computer Science");
        dept.addProffesor(new Professor("Mr James", "Java"));
        dept.addProffesor(new Professor("Mr Scott", "Python"));
        dept.addProffesor(new Professor("Mr Magic", "AI"));

        System.out.println("The professors which are in :"+dept.getDepartmentName()+"
 department");
        dept.getProfessors().forEach(System.out::println);

        Department civil = new Department("Civil Engineering");
        civil.addProffesor(new Professor("Mr Alen", "Engineering Drawing"));
        civil.addProffesor(new Professor("Mr John", "UML"));
        civil.addProffesor(new Professor("Mr Scott", "Drawing"));

        System.out.println("The professors which are in :"+civil.getDepartmentName()+"
 department");
        civil.getProfessors().forEach(System.out::println);


     }
}
```

```java
package com.ravi.arraylist;


import java.util.ArrayList;

public class ArrayListDemo8
{
    public static void main(String[] args)
    {
        ArrayList<String> original = new ArrayList<>();
        original.add("BCA");
        original.add("MCA");
        original.add("BBA");

        System.out.println("By using clone Method");
        @SuppressWarnings("unchecked")
       ArrayList<String> cloned = (ArrayList<String>) original.clone();
        System.out.println(cloned);

        System.out.println("By using Copy Constructor");
        ArrayList<String> copy = new ArrayList<>(original);
        System.out.println(copy);


    }
}
```

---

public List subList(int fromIndex, int toIndex) :

It is used to fetch/retrieve the part of the List based on the given index available inb List interafce. The return type of this method is List, Here fromIndex is inclusive and toIndex is exclusive.

public boolean contains(Object element) :

It is used to find the given element object in the corresponsing List, if available it will return true otherwise false.
It is a method of Collection interface.

public default boolean removeIf(Predicate<T> filter) [JDK 1.8]

It is used to remove the elements based on boolean condition passed as a Predicate.It is available in Collection interafce.

```java
package com.ravi.arraylist;


import java.util.ArrayList;
import java.util.List;

public class ArrayListDemo9 {

    public static void main(String[] args)
    {
        ArrayList<Integer> list = new ArrayList<>();
         list.add(1);
         list.add(2);
         list.add(3);
         list.add(4);
         list.add(5);
         list.add(6);
         list.add(7);
         list.add(8);
         list.add(9);
         list.add(10);

         //public List subList(int fromIndex, int toIndex)
         List<Integer> subList = list.subList(2, 5);
         System.out.println(subList);

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

         //public boolean contains(Object obj)
         boolean contains = list.contains(9);
         System.out.println(contains);

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

         //public int indexOf(Object obj)
         System.out.println(list.indexOf(2));

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

         //public void removeIf(Predicate<T> p)
         list.removeIf(num -> num%2==1);
         list.forEach(System.out::println);

    }
}
```

---

public void trimToSize() :

Used to reduce the capacity equal to the current size.
After trim operation It is dynamically Growable.

public void ensureCapacity(int minCapacaity):

Increase the capacity of the ArrayList to avoid frequent resizing.

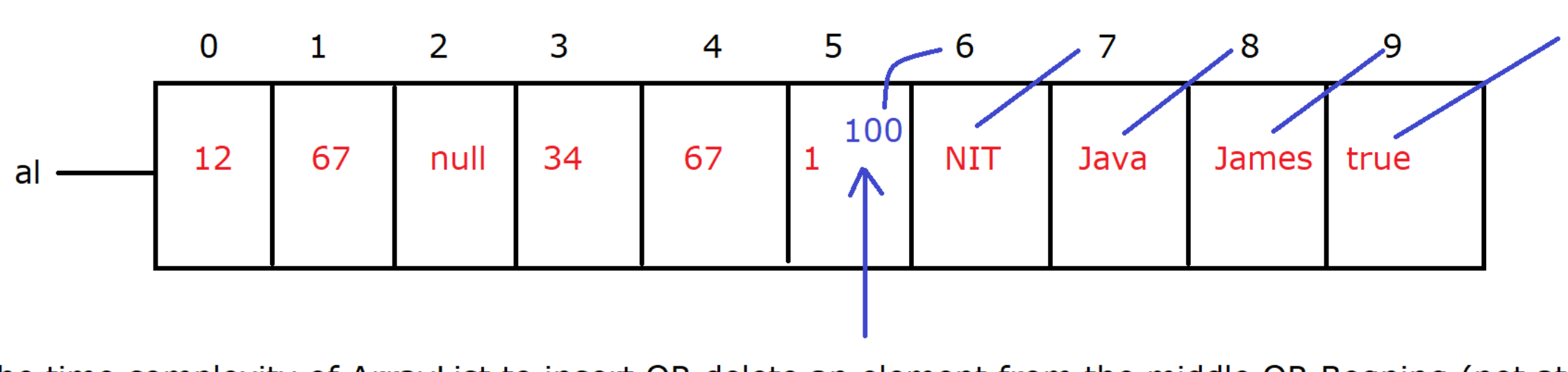
The minCapacaity() method parameter will specify that ArrayList will definetly hold the number of elements specified in the parameter of ensureCapacity() method.

After using ensureCapacity() method, still it is dynamically growable.

```java
package com.ravi.arraylist;


import java.util.ArrayList;

public class ArrayListDemo10 {

    public static void main(String[] args)
    {
        ArrayList<String> list = new ArrayList<>(100);
        list.add("Java");
        list.add("World");


        //public void trimToSize()
        list.trimToSize();
        System.out.println("Trimmed List Size: " + list.size());

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


        ArrayList<Integer> listOfNumber = new ArrayList<>();

        // public void ensureCapacity(int minCapacity)
        //Increase the capacity of the ArrayList to avoid frequent resizing.
        listOfNumber.add(999);

        listOfNumber.ensureCapacity(100);

        for (int i = 0; i < 50; i++)
        {
            listOfNumber.add(i);
        }

        System.out.println("List size: " + listOfNumber.size());


    }

}
```

Time complexity of ArrayList :

ArrayList<Object> al = new ArrayList<>();



The time complexity of ArrayList to insert OR delete an element from the middle OR Begning (not at last) would be O(n) [Big O of n] because 'n' number of elements will be re-located so, it is not a good choice to perform insertion and deletion operation in the middle OR begning of the List.

On the other hand time complexity of ArrayList to retrieve an element from the List would be O(1) because by using get(int index) method we can retrieve the element randomly from the list. ArrayList class implements RandomAccess marker interface which provides the facility to fetch the elements Randomly.

In order to insert and delete the element in middle Or anywhere in the list frequently, we introduced LinkedList class.

*** What is linked list data structure ?
----
* It is a linear OR sequential data structure which stores the element in the form of Node object. Actually we can say linked list is Collection of node object.

* Here each node is inter-linked with another node in a **non contiguous memory location.**

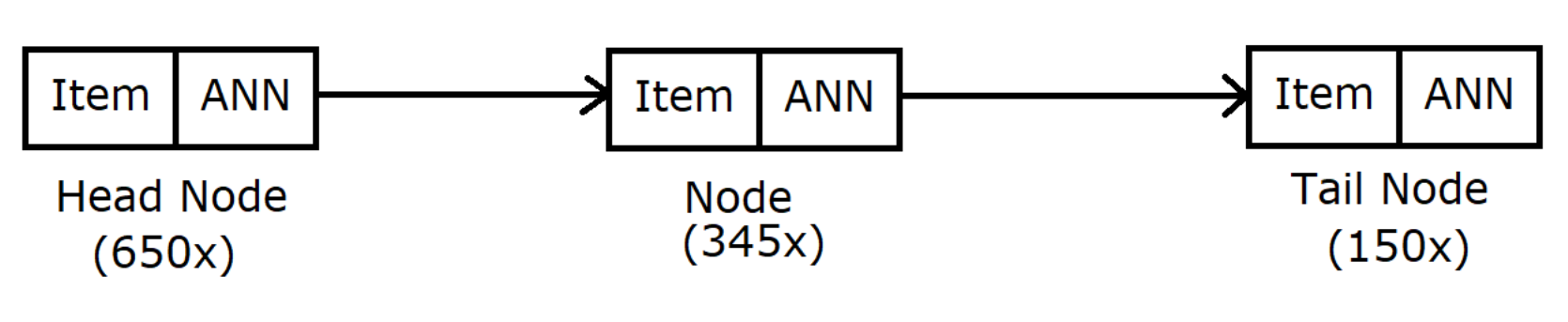
* This linkage facility provides efficient insertion OR deletion anywhere in the list.

We have 3 types of linked list :

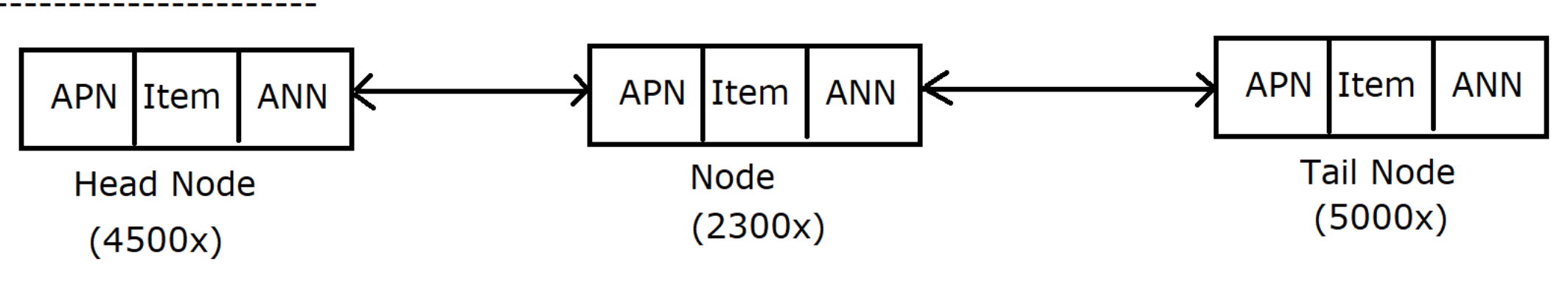
ANN = Address of next node
APN = Address of previous node

1) Singly linked list [In forward direction only]
2) Doubly linked list [In forward + backward direction]
3) Circular linked list [Head node will hold the address of Tail node and vice versa]

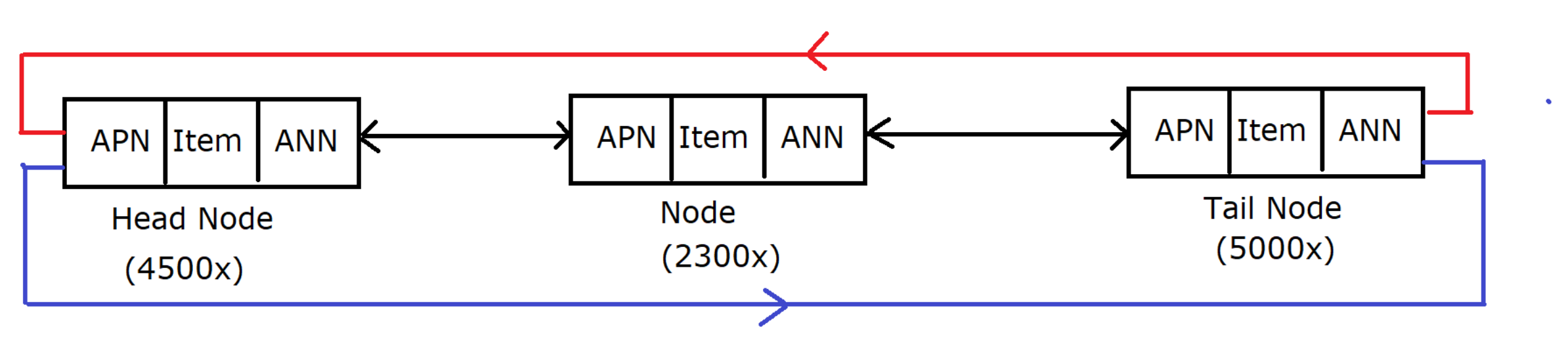
1) Singly linked list :



2) Doubly linked list :

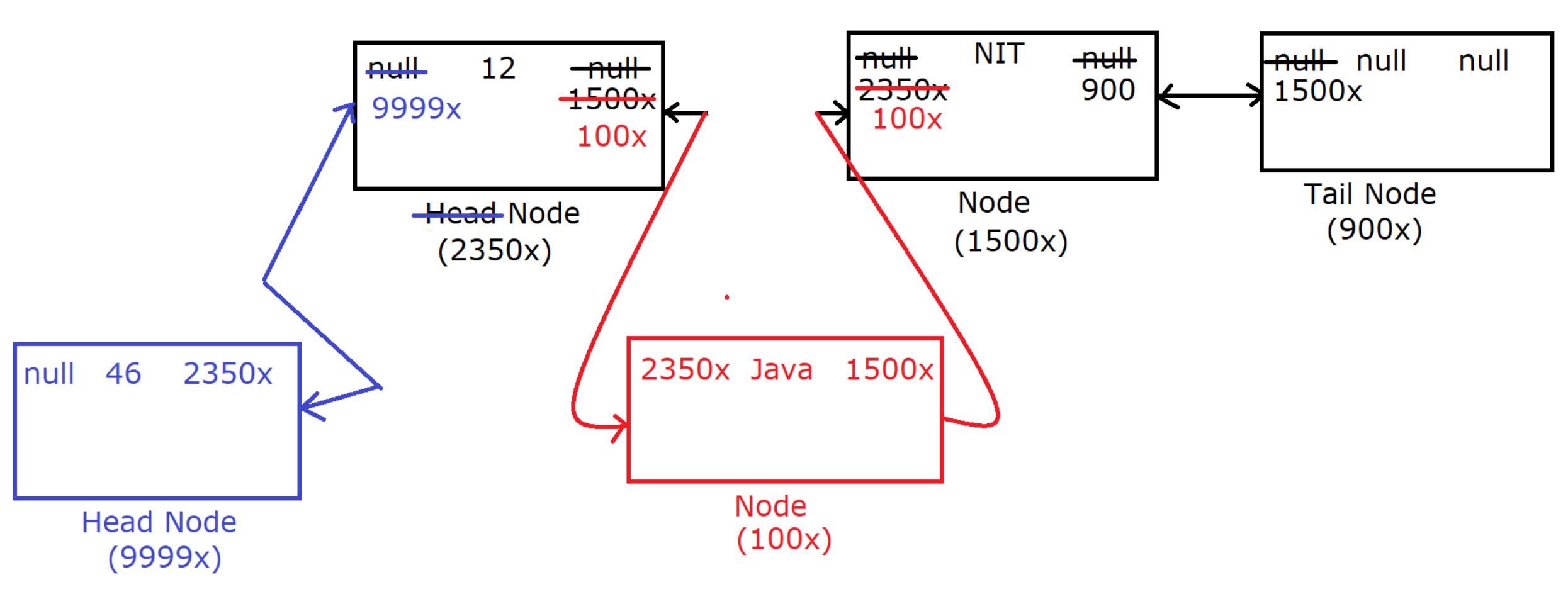


3) Circular linked list :



Implemenation of Doubly LinkedList in java :

```
class Node              //Node is private static inner class
{
   E item;
   Node next; //ANN
   Node prev; //APN

   Node new= new  Node(prev, E, next)
   {

   }
}


LinkedList<Object> list = new LinkedList<>();
list.add(12);
list.add("NIT");
list.add(null);
list.add(1, "Java");
list.addFirst("46");
```



Singly Linked List Custom Implementation :

```java
package com.ravi.linked_list;

public class CustomLinkedList
{
    private static class Node
    {
        int item;
        Node next;  //ANN [Address of Next Node]

        public Node(int item)
        {
            this.item = item;
            this.next = null;
        }
    }

    public static void main(String[] args)
    {
        Node n1 = new Node(100);
        Node n2 = new Node(200);
        Node n3 = new Node(300);

        n1.next = n2;
        n2.next = n3;

        System.out.println(n1.item);
        System.out.println(n1.next.item);
        System.out.println(n1.next.next.item);
    }

}
```