

Interface ==> contains abstract methods(methods without any body)
Built in interfaces in java ==> already present in JAVA

1) List Interface

It is a interface that is already existed in java, that basically contains 8 to 10 abstract methods.

```
interface List {  
    public void add();  
    public void remove();  
    .....  
}
```

A class is the one that makes the interface methods work!

ArrayList is a class that basically implements List interface

When to use ArrayList?

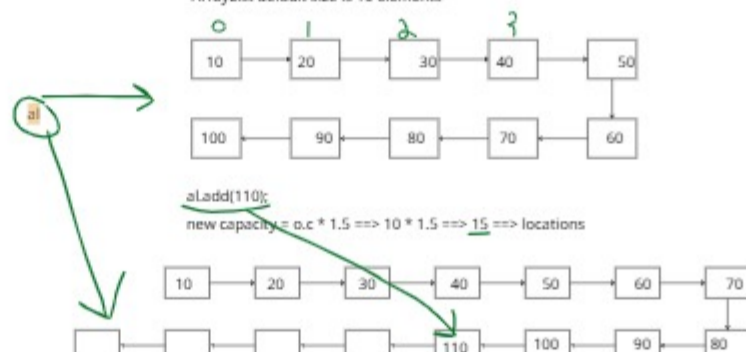
There are 2 problems in Arrays, to overcome those problems we use ArrayList

- 1) Arrays store same type of data
- 2) Arrays size is fixed
`int[] marks = new int[5];`
5 marks

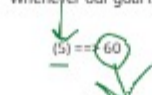
I want to store different types of data and i also don't know how many elements i want to store ==> ArrayList

Arrays ==> fixed
ArrayList uses a data structure ==> **Dynamic Array**
ArrayList internally uses Arrays only to store the data but those arrays are Dynamic(size can change)

ArrayList default size is 10 elements



Whenever we want to insert elements to the ArrayList ==> Worst Choice
Whenever our goal is reading ==> Best Choice



Duplicate elements are allowed in ArrayList
Insertion order is maintained in ArrayList



```
compareTo()  
{  
    .....  
}  
}
```

java ==> Comparable Interface ==> contains compareTo()

Interface ==> abstract method(without body)

Comparable(l) ==> compareTo() { - }

java 8 onwards ==> Interface ==> abstract methods + concrete methods

Comparator ==> Interface ==> helps you to implement multiple sorting logic based conditions

Comparable ==> compareTo()
Comparator ==> compare()

arrayList ==> { 1, 5, 3, 2, 4 }

target = 2

print the total no of pairs that gives the target
| a b | = target

(1, 3), (5, 3), (2, 4) ==> 3 pairs

arrayList ==> { 1, 5, 3, 2, 4 }

target = 4

| a b | = target

a + b = target

AND

(a - b) = target

a + b = target

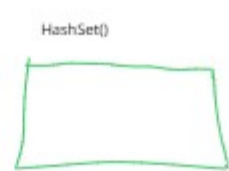
a = target - b

a = target + b

a = b - target

a - b = target

a = b + target



firstNum - secondNum = target

firstNum - secondNum = target

secondNum = firstNum - target

secondNum = firstNum + target

{ 10, 15, 13, 12, 14 }

target = 3

firstNum = 13

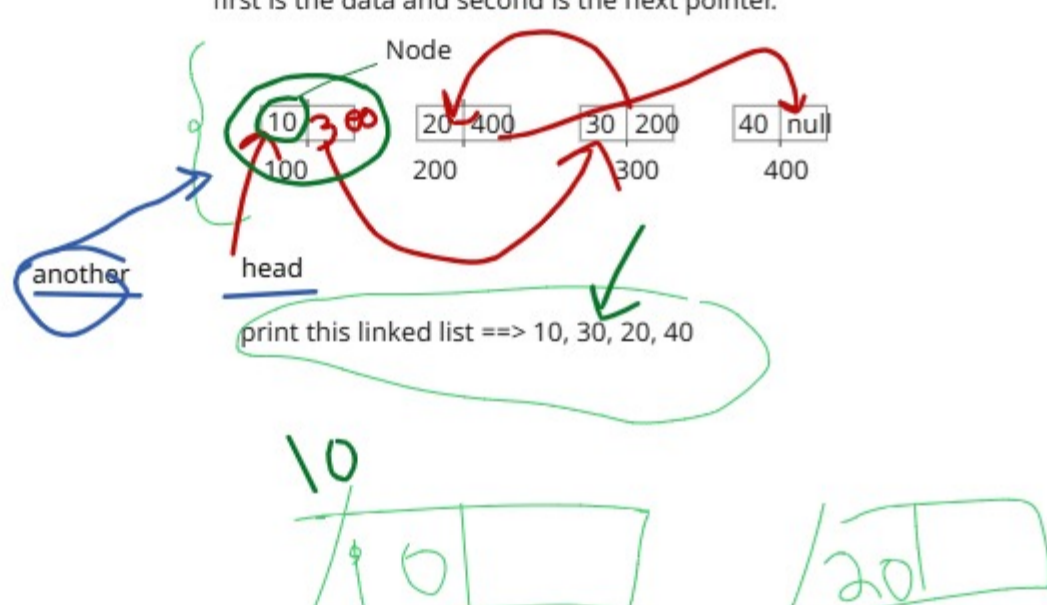
secondNum = 10

Linked List

Linked List ==> uses doubly linked list as a data structure to store the data

singly linked list

In SLL, we have nodes and each node has 2 parts, first is the data and second is the next pointer.



```
Node firstNode = new Node(10);  
Node secondNode = new Node(20);
```

LinkedList is best choice if our goal is insertion operation

LinkedList is worst choice if our goal is read operation

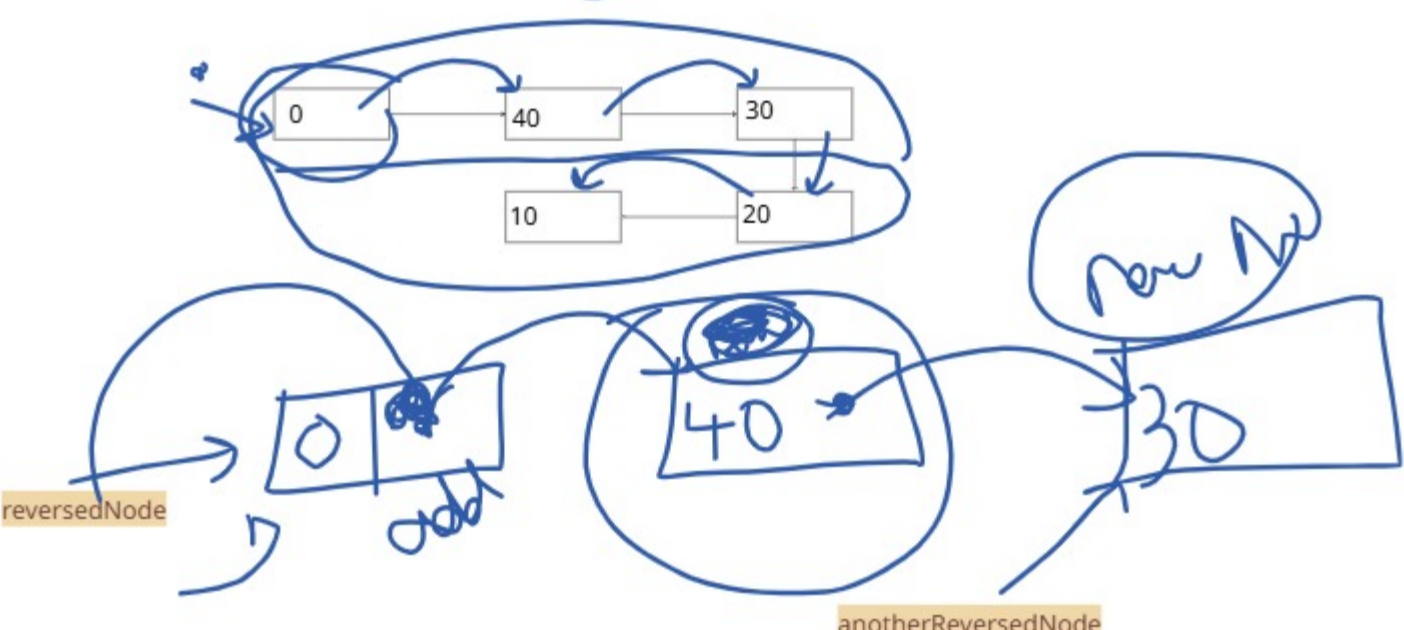
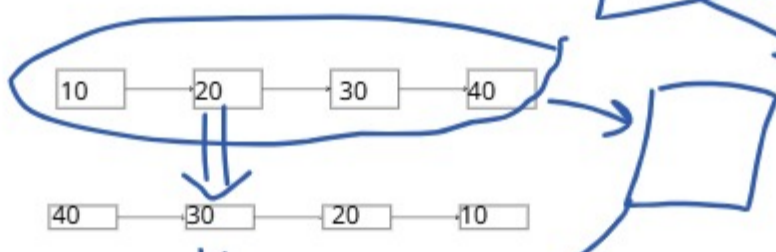


1000 ==> between 10 and 30

Reverse a singly linked list

10 ==> 20 ==> 30 ==> 40
40 ==> 30 ==> 20 ==> 10

```
while(anotherHead != null)  
{  
    System.out.print(anotherHead.data+ " ==> ");  
    anotherHead = anotherHead.addr; // anotherHead = 300  
}
```



reversedNode.addr ==> 2nd node(40)