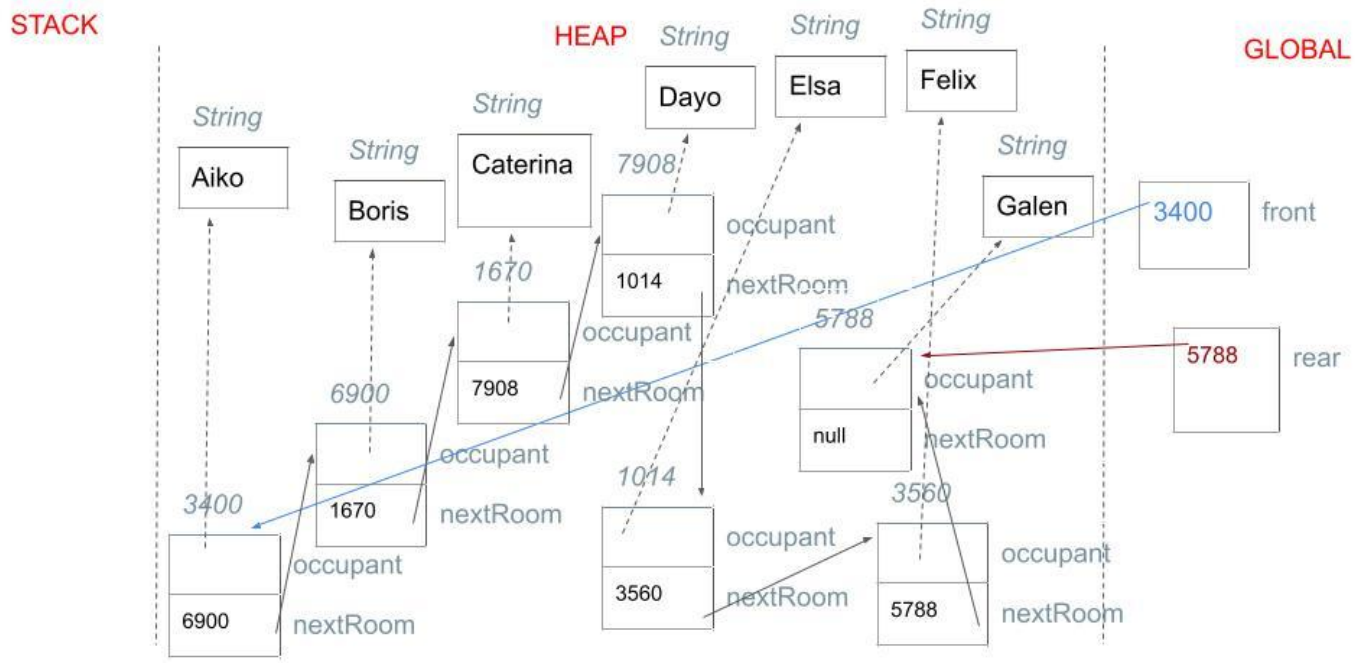
Dmitri Stanchevici
Unit 4
Module 2

Ex. 2.1

Conceptual memory picture for the linked list just after makeList() completes.



Ex. 2.4

Tracing the values of pointer and prePointer through the while iterations, using the addresses from ex. 2.1.

1. Before entering the while loop:
   pointer = 3400
2. In while loop

iteration	**pointer**
1	6900
2	1670
3	7908
4	1014
5	3560

1. Before entering the while loop:
   prePointer = 3400
2. In while loop

iteration	**prePointer**
1	6900
2	1670
3	7908
4	1014

Ex. 2.5

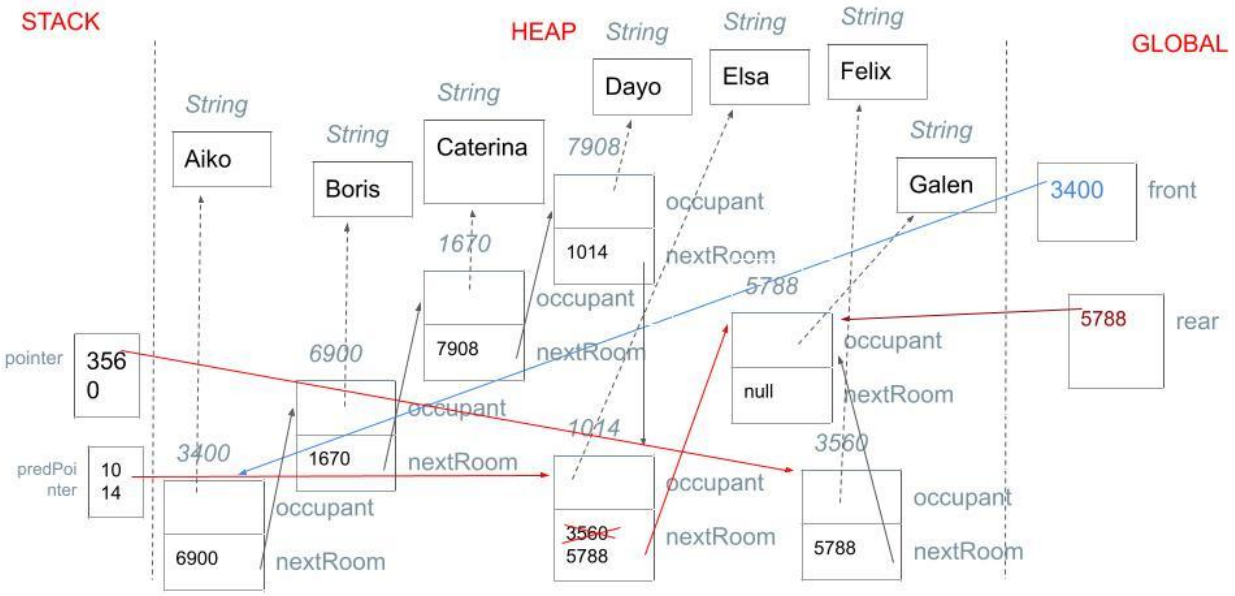
Changing the name to “Aiko” in main() throw the following error:

```
Exception in thread "main" java.lang.NullPointerException: Cannot read field "nextRoom" because "<local2>" is null
        at Hotel2.predecessorPrint(Hotel2.java:27)
        at Hotel2.main(Hotel2.java:13)
```

This happens because the condition prePointer.nextRoom != pointer in the while loop is never satisfied. When the loop has finished traversing the list, it bumps into the last Room object whose nextRoom value is null.

Ex. 2.7

Memory picture right after the completion of remove()



Ex. 2.8

Yes, the program works fine for removing the last name: Galen.

```
dmitristanchevici@Dmitris-iMac module2 % java Hotel3
Aiko
Boris
Caterina
Dayo
Elsa
Felix
```

Ex. 2.9

“Galen” is removed with remove(), but “Hector” is not added with addToList() because rear continues to point to the object containing “Galen.” So, it is not linked to the rest of the list that is printed with printList() starting with front.

To fix this problem, I added the following at the end of remove() to reassign the last element to rear:

```
if (predPointer.nextRoom == null) {
    rear = predPointer;
}
```

Ex. 2.11

1. Before entering the while loop:
    pointer = 3400
    predPointer = 3400
2. In while loop

iteration	**predPointer**	**pointer**
1	3400	6900
2	6900	1670
3	1670	7908
4	7908	1014
5	1014	3560

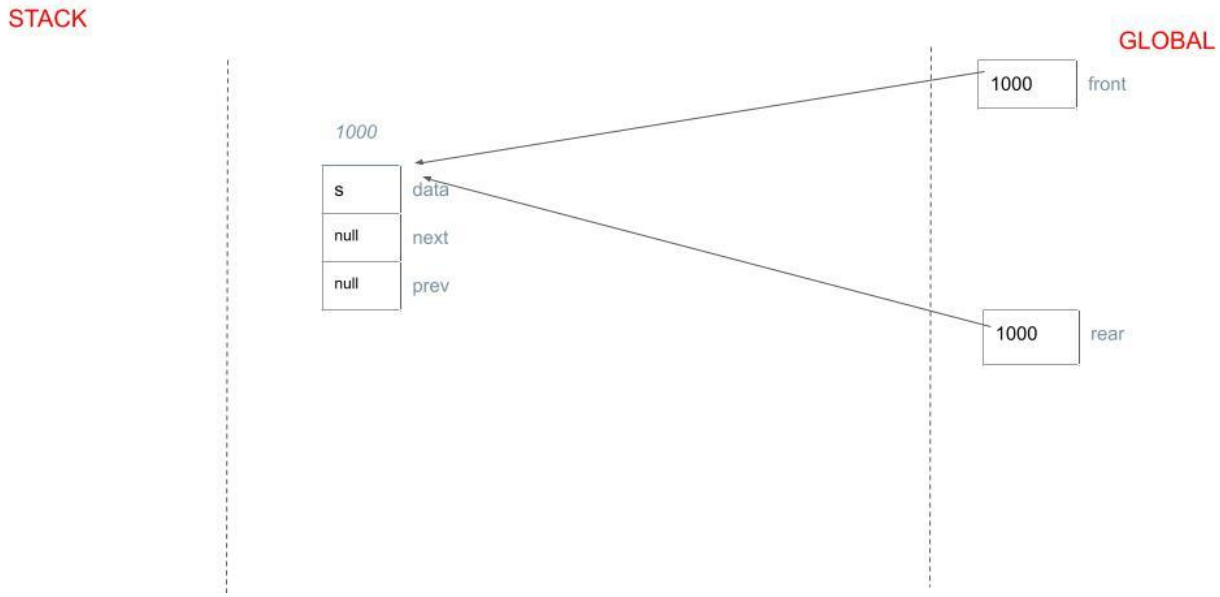
3. Break out of the while loop

4. predPointer.nextRoom = pointer.nextRoom (predPonter.nextRoom = 5788).

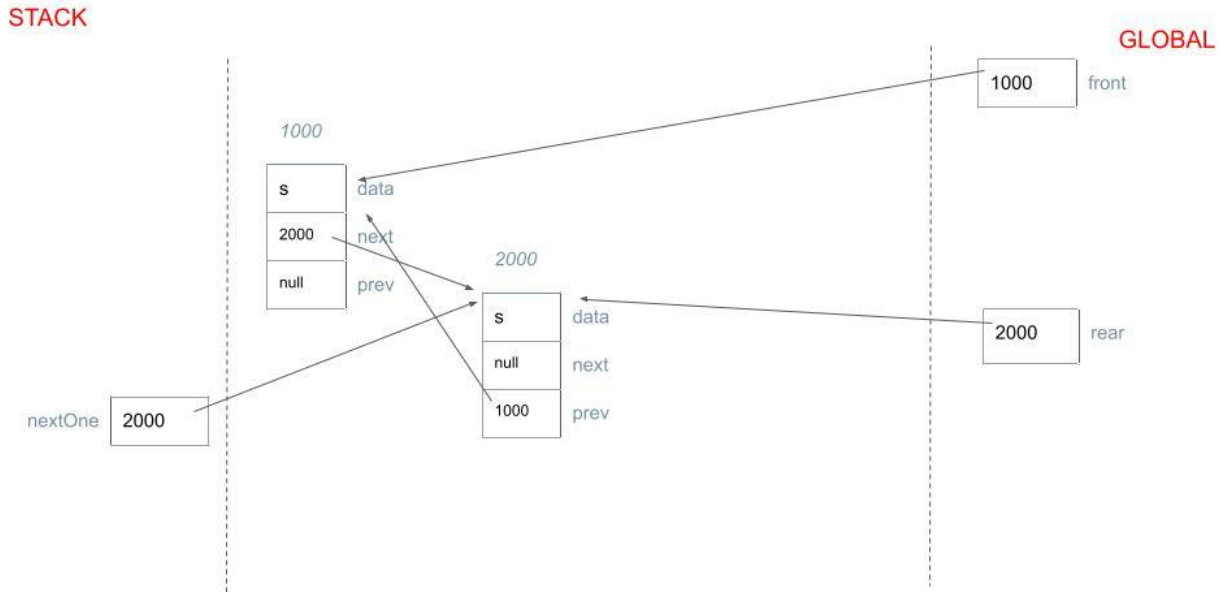
Ex. 2.16

Conceptual pictures of memory in a double-linked list with three elements added to empty list

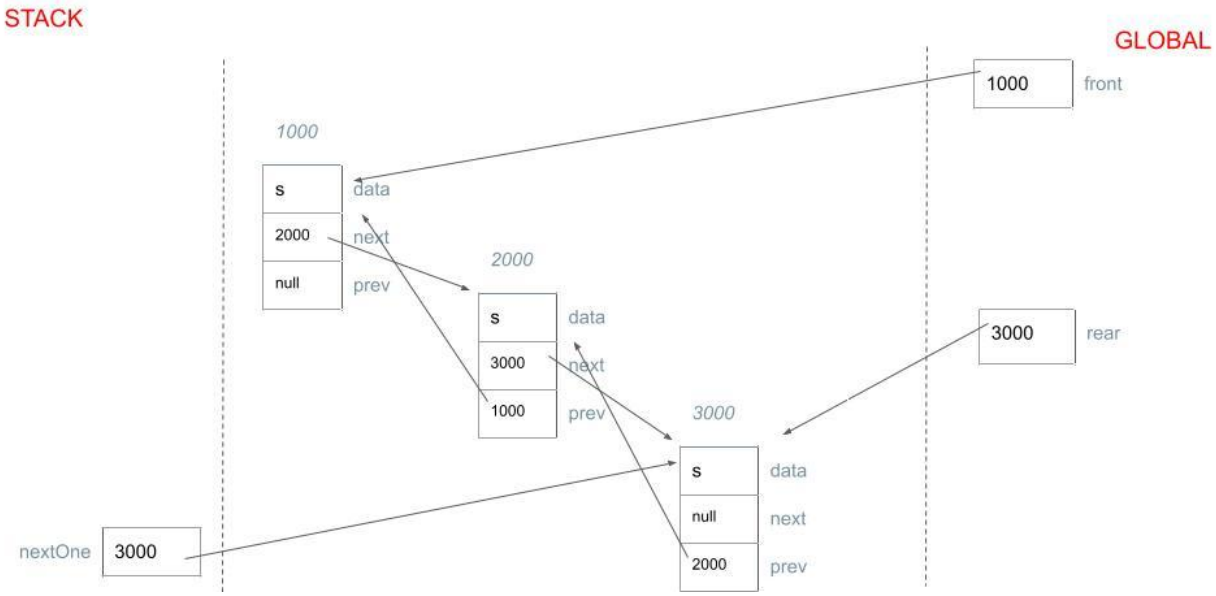
1 Add first element to empty list.



2 Add second element.



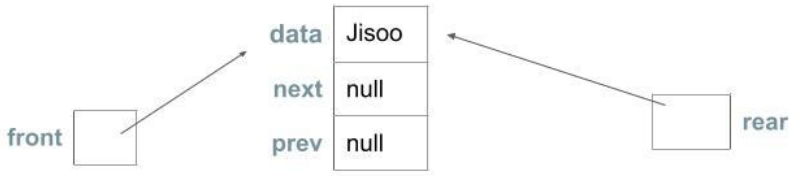
3 Add third element.



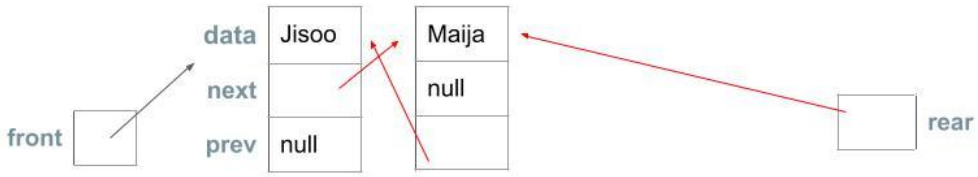
Ex. 2.22

Conceptual pictures of a list sorted in alphabetical order, illustrating how links change after each addition to the list

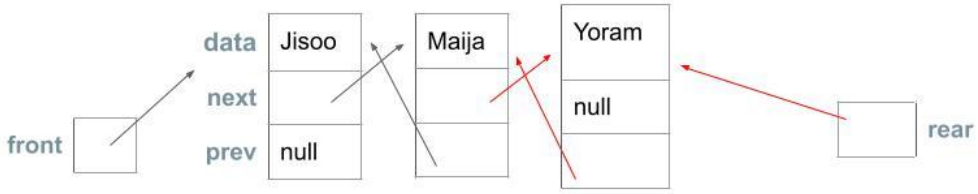
Step 1



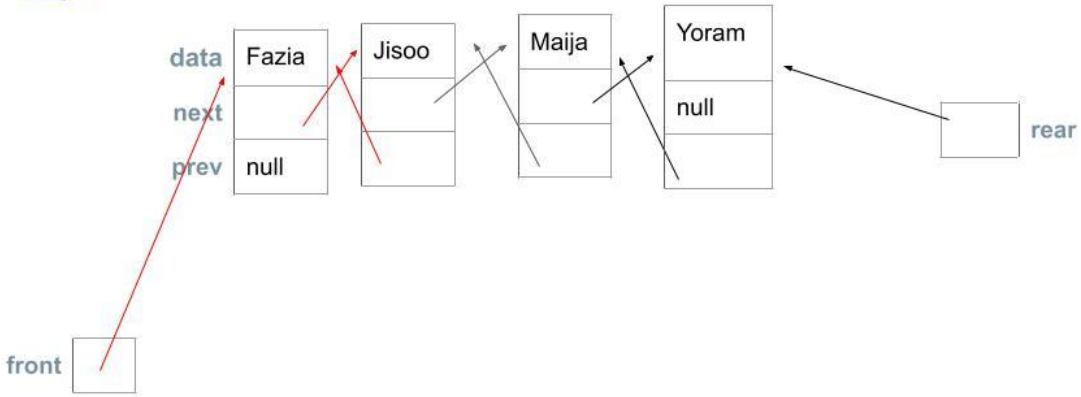
Step 2



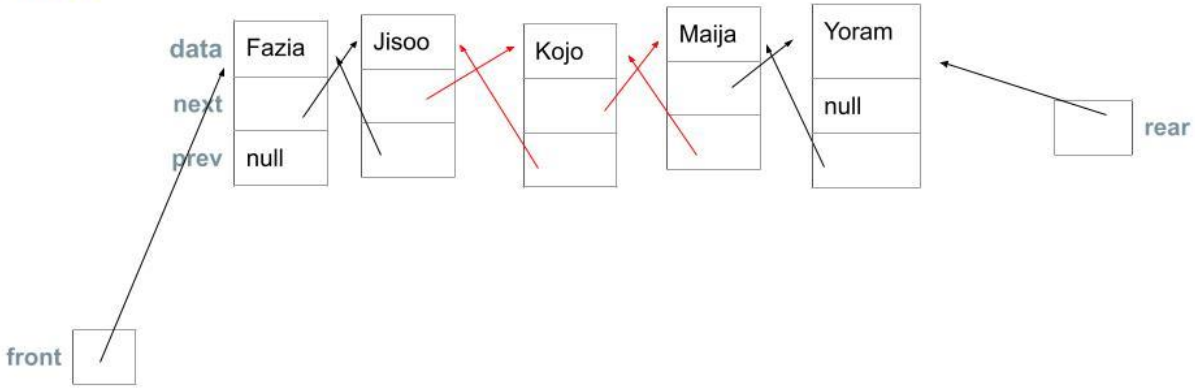
Step 3



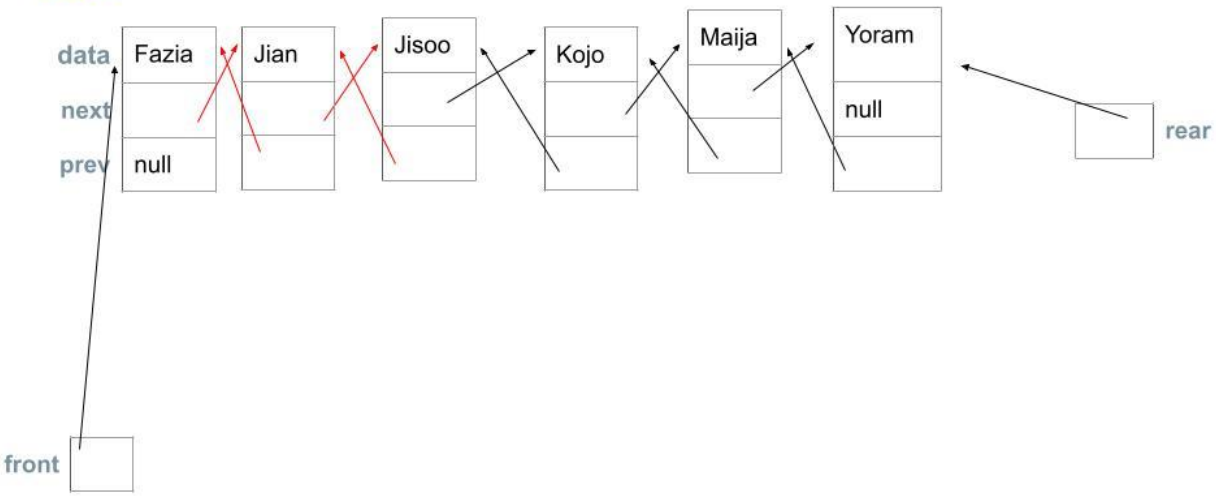
Step 4

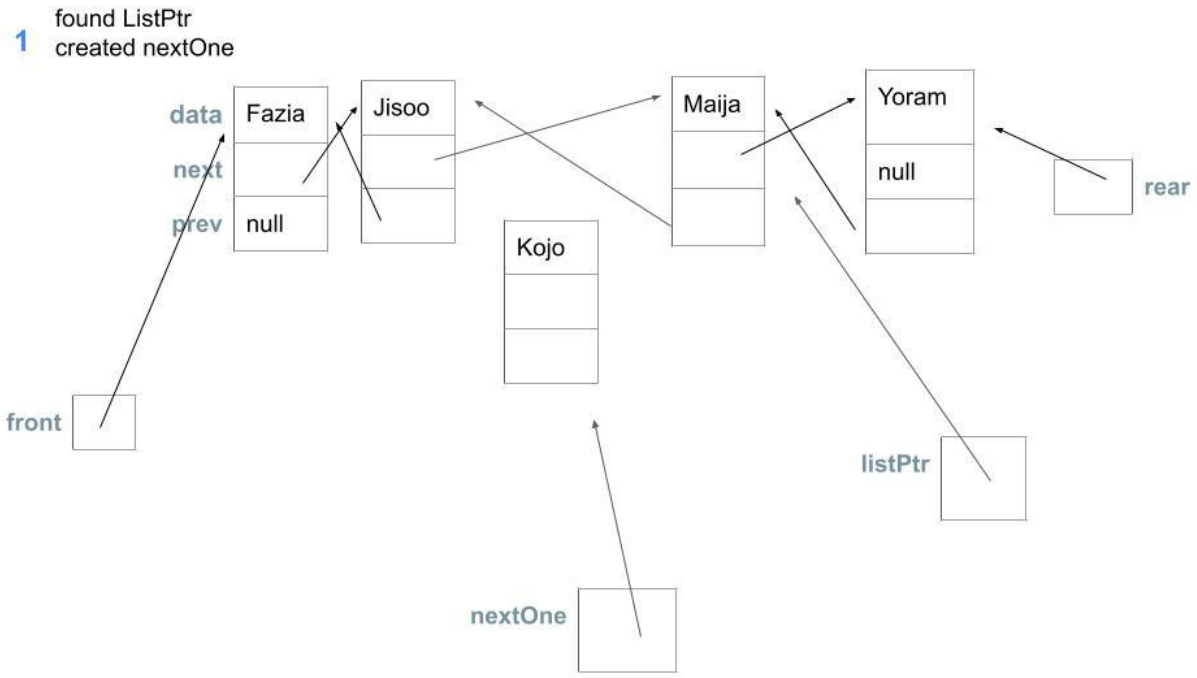


Step 5



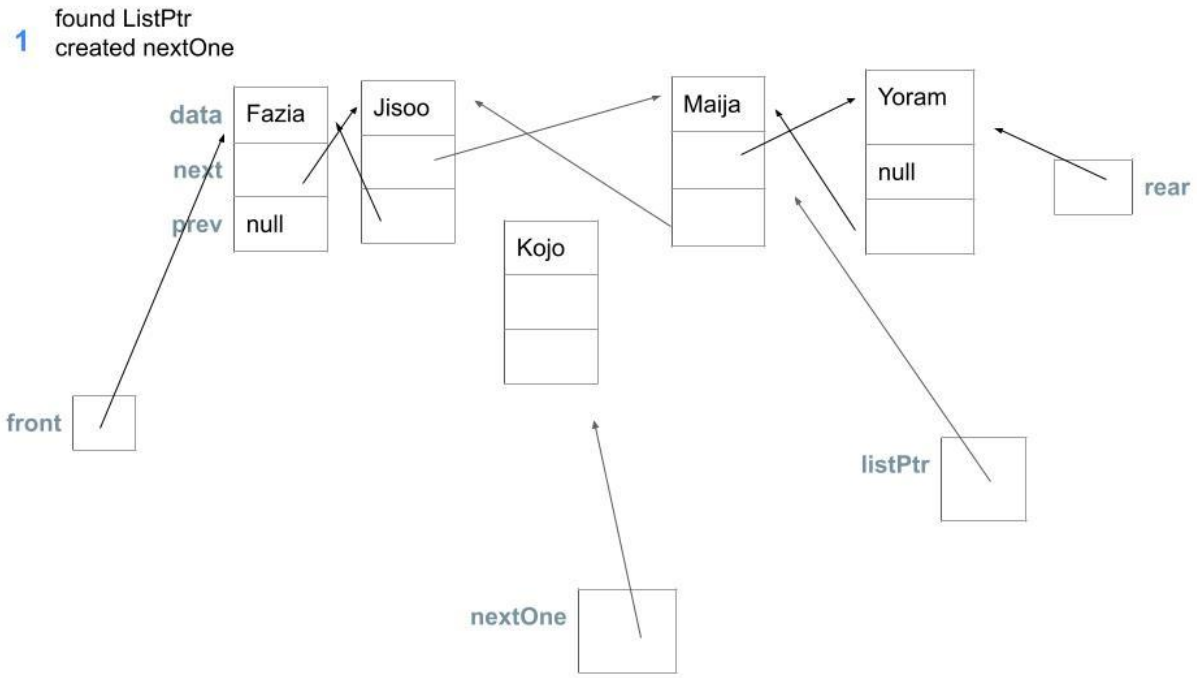
Step 6



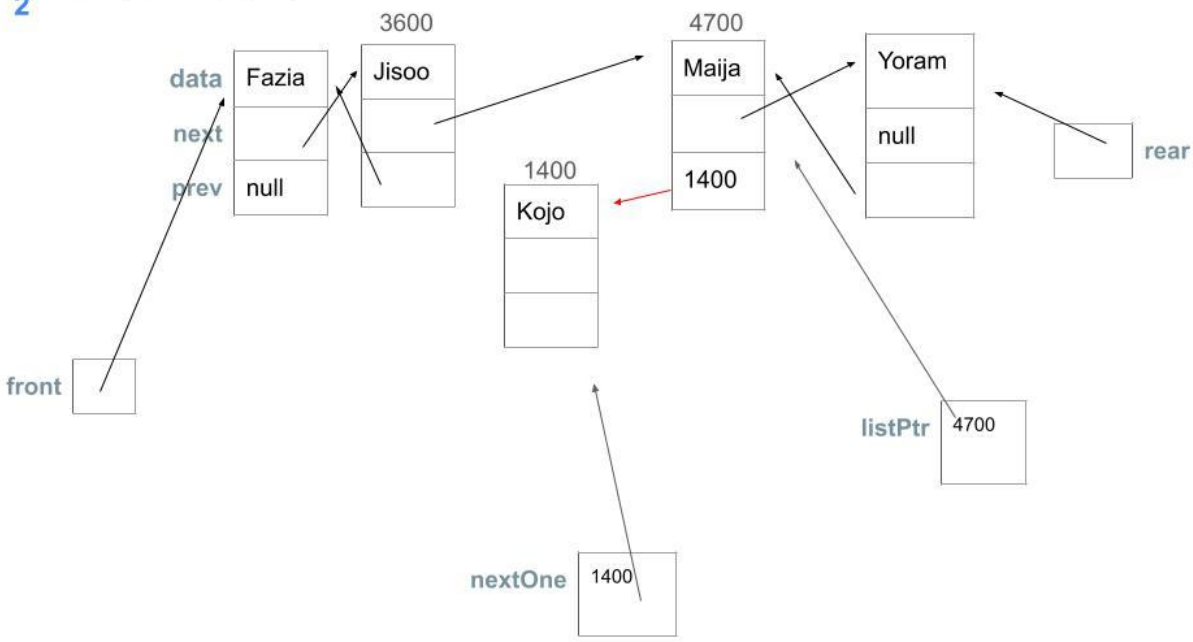


Ex. 2.24

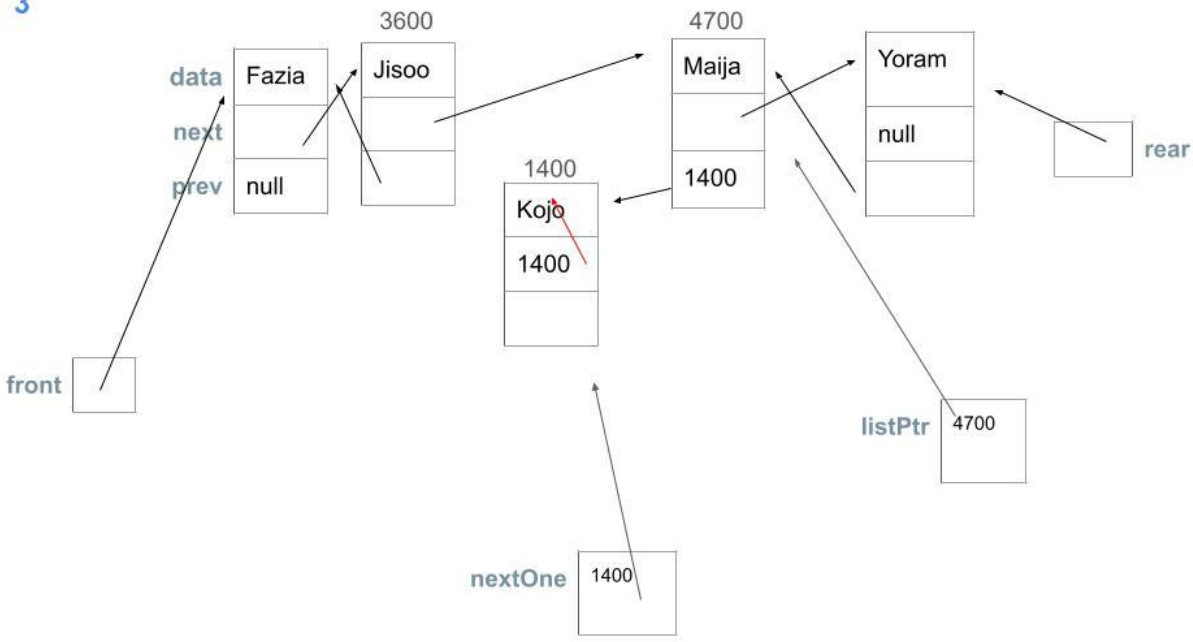
Explaining why the order of link establishment is important. The pictures below show what goes wrong.

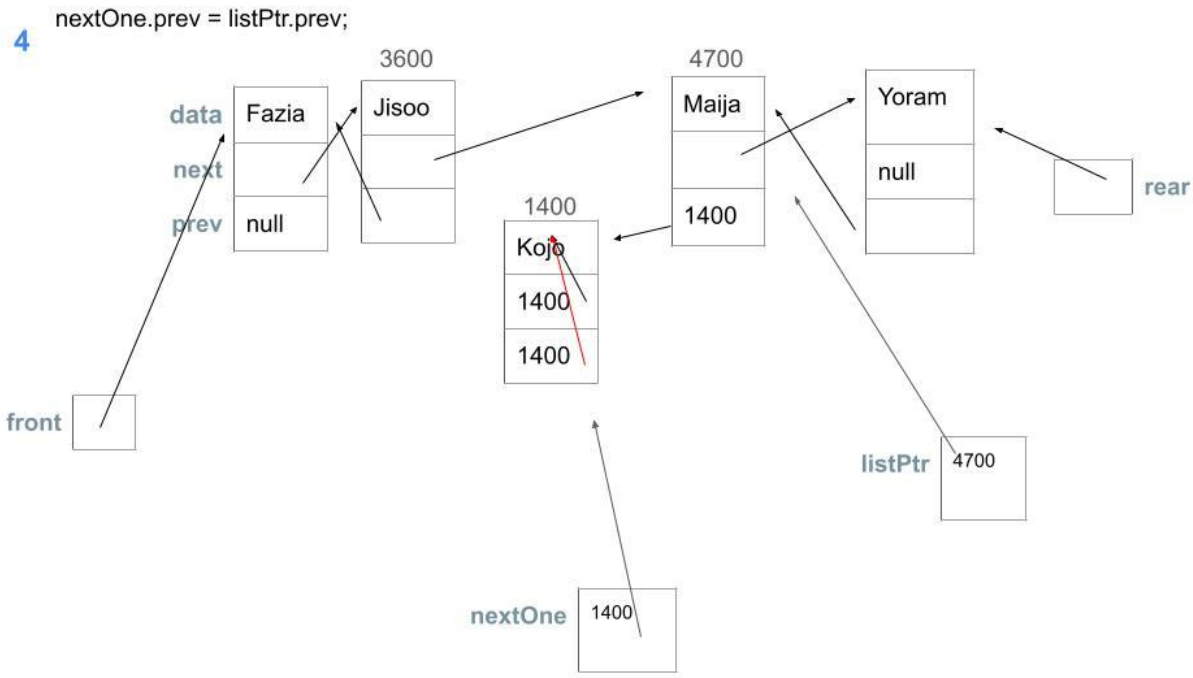


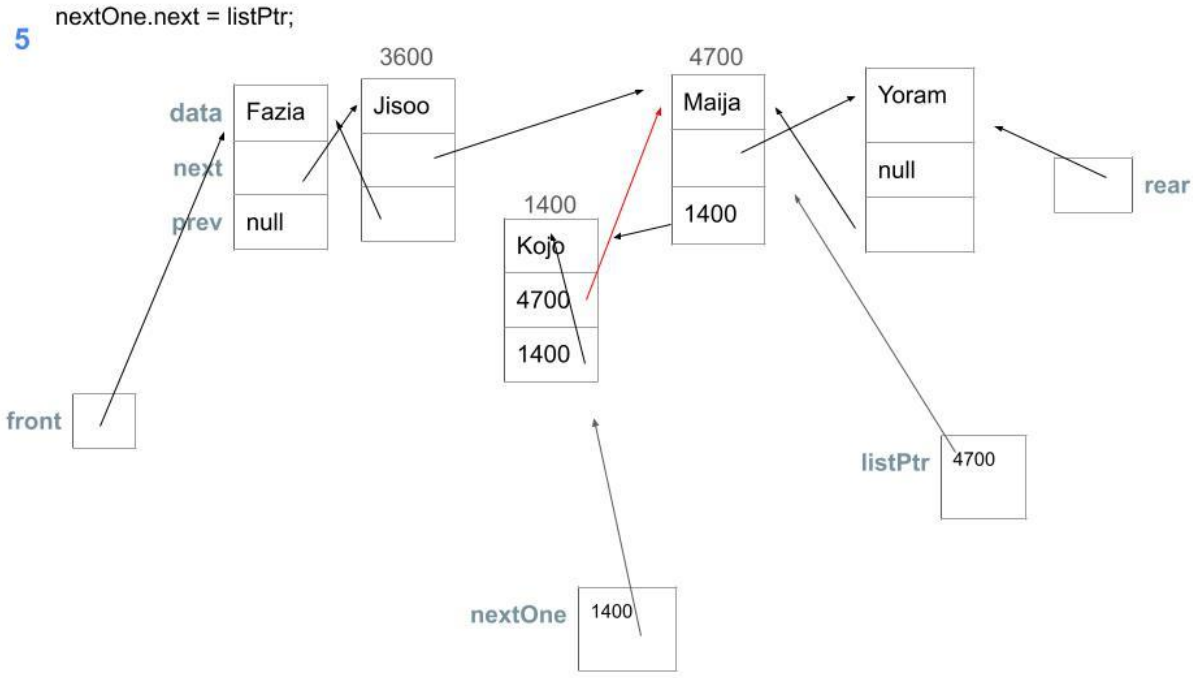
2 listPtr.prev = nextOne;



3 listPtr.prev.next = nextOne;







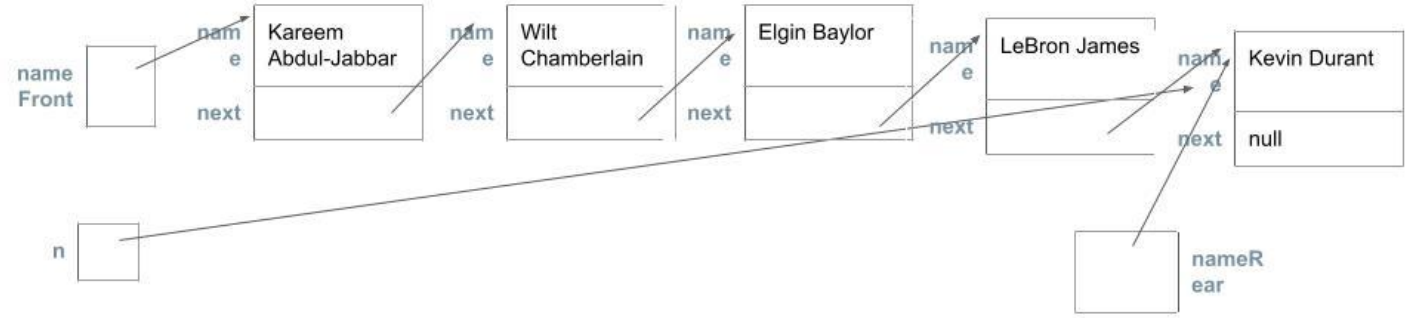
The problem is that **nextOne.prev** points to **nextOne,** instead of to the preceding node. Thus, because there is no connection between “Jisoo” and “Kojo,” but there is one between “Jisoo” and “Majia,” **printList()** skips “Kojo,” printing:

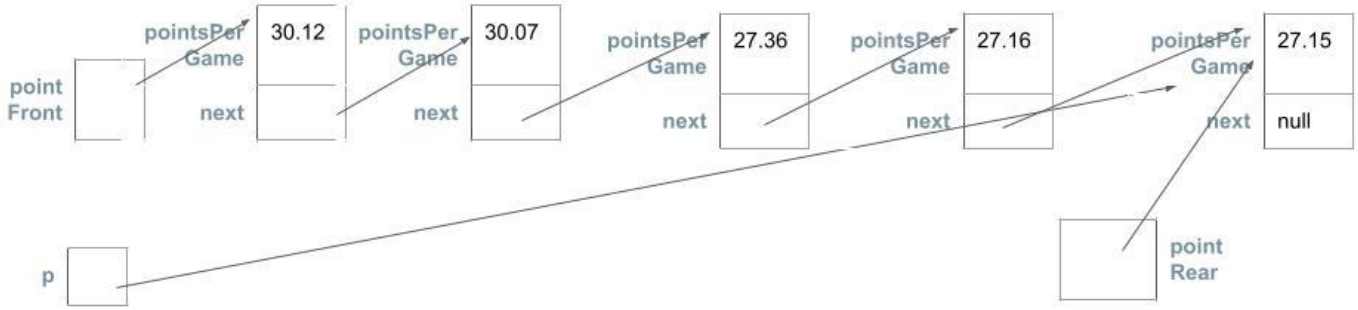
[Fazia Jisoo Majia Yoram]

printReverse() has another problem. It gets to “Kojo” but then circles back to it because the **prev** in Kojo points to Kojo. The line **listPtr = listPtr.prev** in the while loop never allows **listPtr** to become null and get out of the loop, thus making this loop infinite.

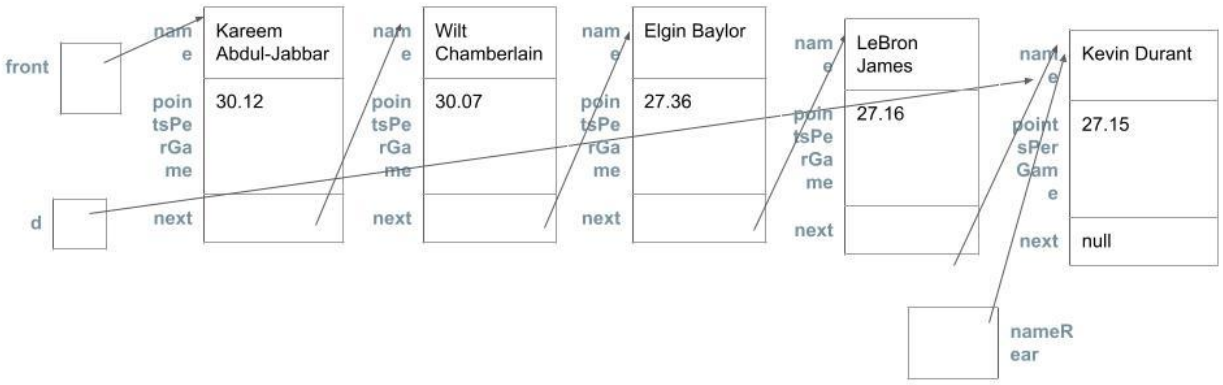
Ex. 2.26

Picture of memory after all nodes have been added in **WrongMultiData.java**.





Ex. 2.27



Ex. 2.32

The program did not run. Here is the feedback:

```
dmitristanchevici@Dmitris-iMac module2 % java WrongChemistrySimulation
Exception in thread "main" java.util.ConcurrentModificationException
        at java.base/java.util.LinkedList$ListItr.checkForComodification(LinkedList.java:970)
        at java.base/java.util.LinkedList$ListItr.next(LinkedList.java:892)
        at WrongChemistrySimulation.react(WrongChemistrySimulation.java:60)
        at WrongChemistrySimulation.main(WrongChemistrySimulation.java:31)
```

Ex. 2.33

Here is the output:

```
Iterator time: 3
Get-loop time: 4111
```

Without the get() method, the iterator reaches a node, uses its data, and moves to the next node. The iterator here does NOT need to start traversing the list from the front node on every iteration; it continues to the next node.

The get() method, however, requires traversal from the start of the list to reach the retrieved element. So, in this example, the list goes from the front node to the i-th node on every iteration.