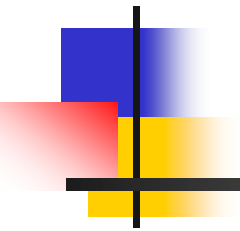


Lecture 8

연결 리스트

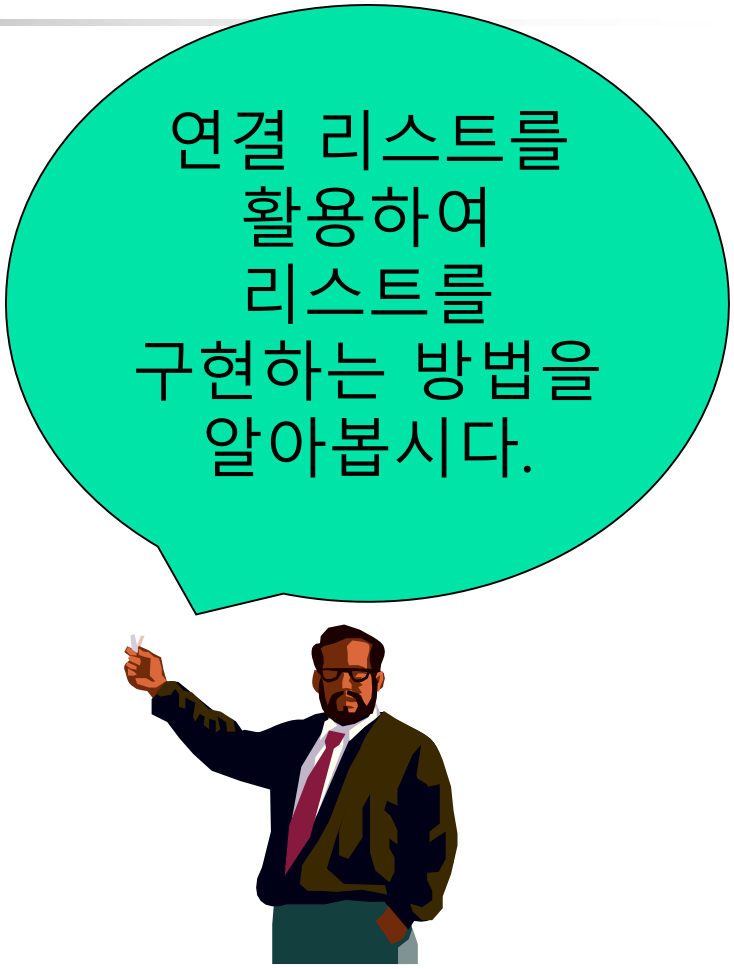
2018년도 2학기

컴퓨터프로그래밍2

김 영 국
충남대학교 컴퓨터공학과

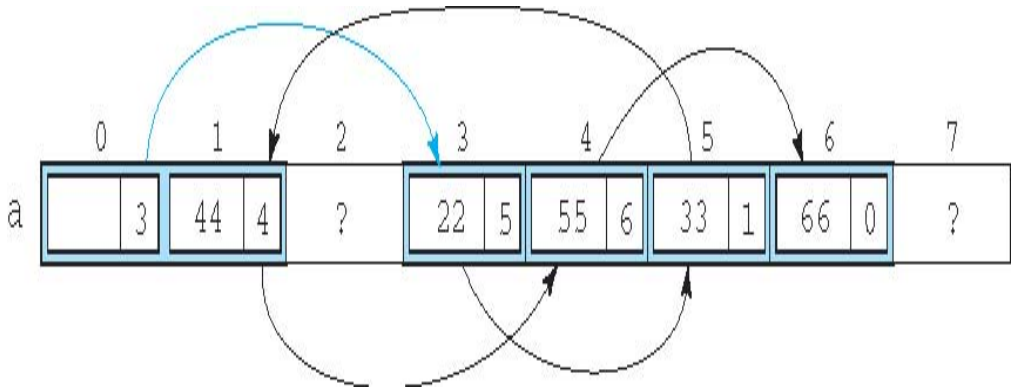
이번 주에 학습할 내용

- 연결 노드
- 연결리스트에 원소 삽입
- 리스트의 앞에 삽입
- 연결리스트에서의 원소 삭제
- 중첩 클래스
- 사례연구: 임의의 긴 정수



1. 연결 노드

- 인덱스 배열 k[]와 데이터 배열 a[]의 상대적인 위치는 일치하므로 이들을 데이터-주소 쌍의 단일 배열로 표현

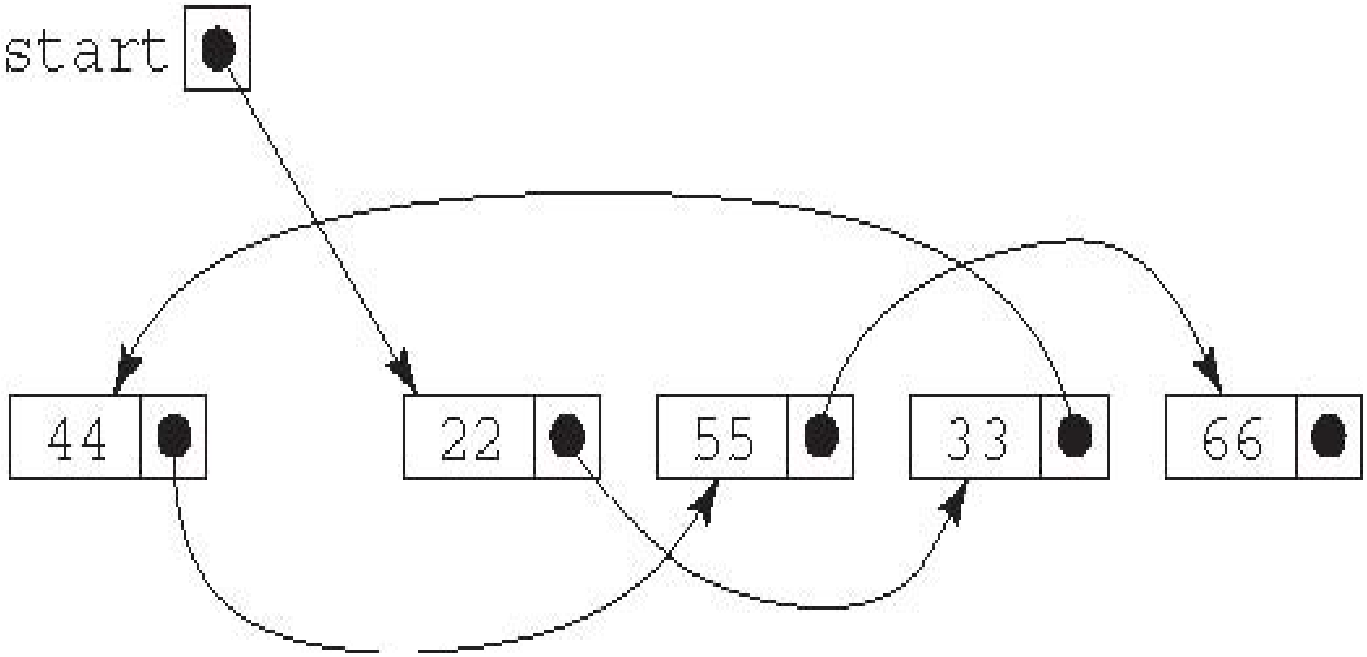


- 배열 a[]의 정의

```
Node[] a = new Node[size];
class Node {
        int data;
        int next;
 }
```

원소에 그 참조를 위한 객체의 사용

- Node 객체의 시퀀스로 생각할 수 있음
  - 배열 a[] 대신에 단일 start 참조만을 유지

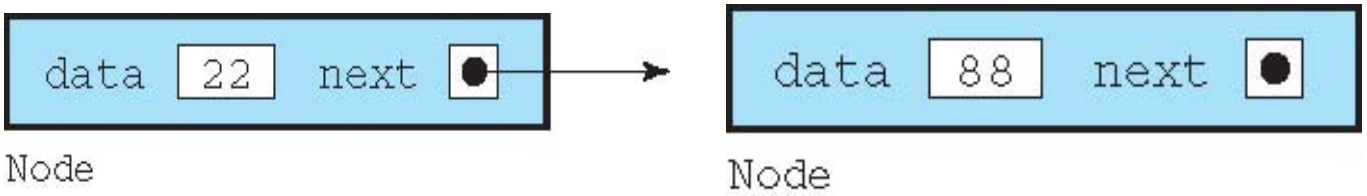


Node 클래스

- Node 클래스는 자기 참조 형태로 정의
  - **LISTING 1: A Node Class**

```
1 class Node {
2     int data;
3     Node next;
5     public Node(int data) {
6             this.data = data;
7     }
8 }
```

- 전형적인 Node 객체



연결 리스트의 생성

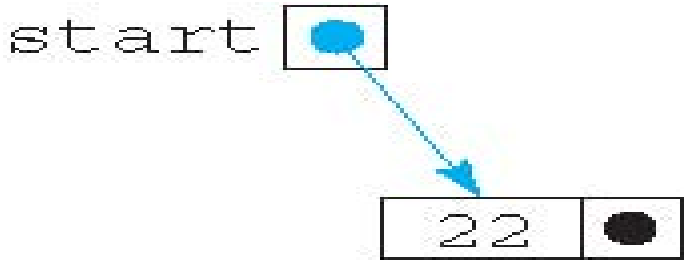
- 5-원소 리스트의 생성
  - **LISTING 2: Constructing a Linked List**

```
1 Node start = new Node(22);
2 start.next = new Node(33);
3 start.next.next = new Node(44);
4 start.next.next.next = new Node(55);
5 start.next.next.next.next = new Node(66);
```

연결 리스트의 생성

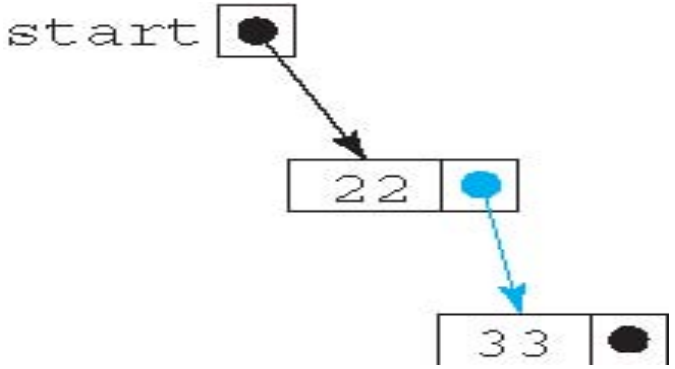
- start의 초기화

```
Node start = new Node(22);
```



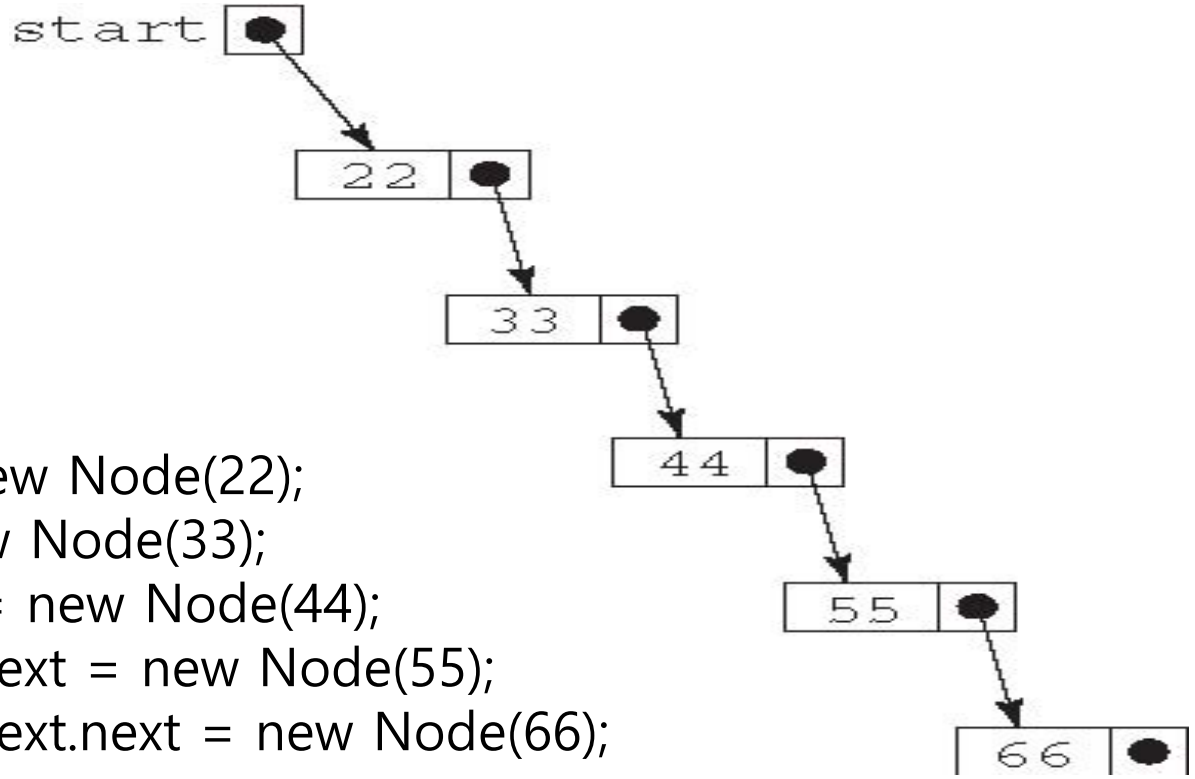
- 한 노드의 추가

```
start.next = new Node(33);
```



연결 리스트의 생성

- 노드 리스트




```
Node start = new Node(22);
start.next = new Node(33);
start.next.next = new Node(44);
start.next.next.next = new Node(55);
start.next.next.next.next = new Node(66);
```

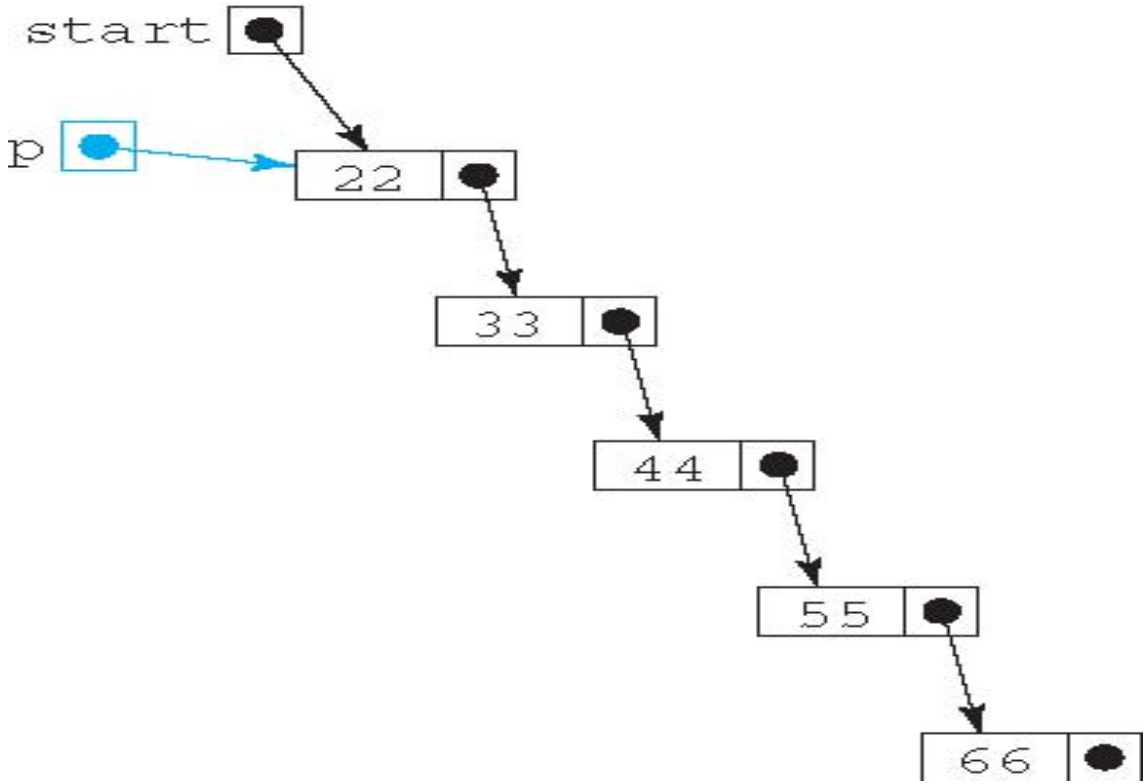
연결 리스트의 생성

- 리스트의 노드들을 순회할 수 있는 지역 참조 변수의 사용
  - 전통적으로 변수 p(포인터를 의미)를 사용
  - 변수 p는 Node 참조로 선언
  - 연결 리스트의 생성
- **LISTING 3: Constructing a Linked List**

```
1 start = new Node(22);
2 Node p = start;
3 p.next = new Node(33);
4 p = p.next;
5 p.next = new Node(44);
6 p = p.next;
7 p.next = new Node(55);
8 p = p.next;
9 p.next = new Node(66);
```

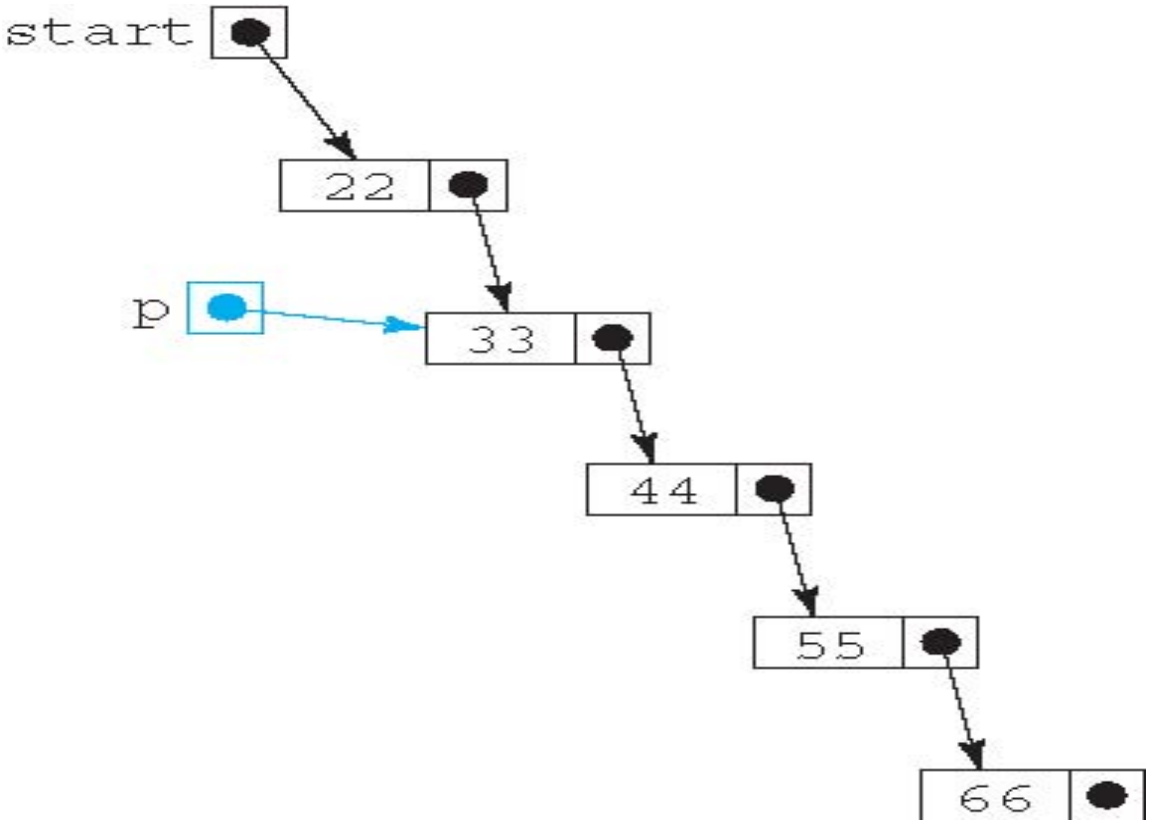
p를 start 노드로 초기화

Node p = start;



p를 두 번째 노드로 전진

p = p.next;

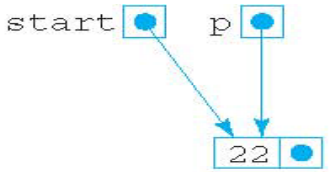


for 루프의 사용

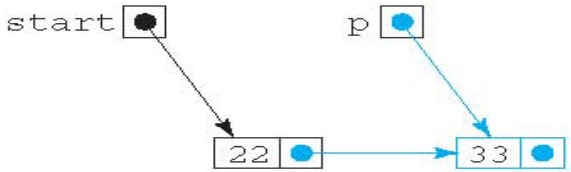
- 루프를 이용한 동일한 연결 리스트의 생성
- **LISTING 4: Using a for Loop**

```
1 Node p = start = new Node(22);
2 for (int i=0; i<4; i++)
3       p = p.next = new Node(33+11*i);

/* 위 2, 3 line과 동일한 의미
2 for (int i=0; i<4; i++) {
3       p.next = new Node(33+11*i);
4       p = p.next;
5  }
*/
```

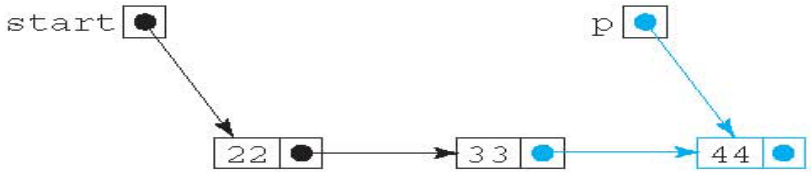
p = start = new Node(22);



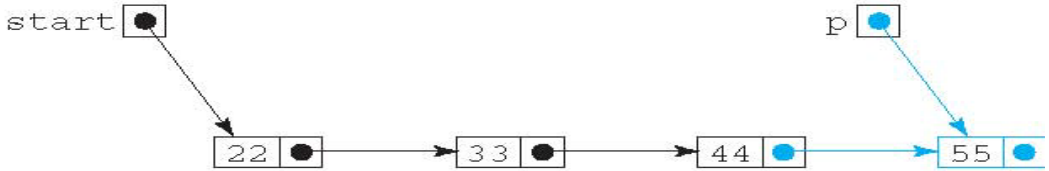
p.next = new Node(33);
p = p.next;



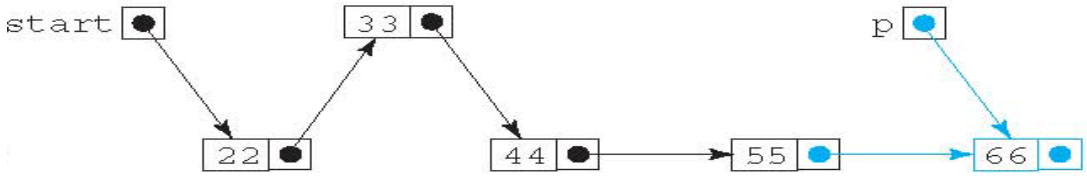
p = p.next = new Node(44);



p = p.next = new Node(55);



p = p.next = new Node(66);



for 루프의 사용

- 루프를 이용한 연결 리스트의 프린팅

LISTING 5: Using a for Loop to Print a Linked List

```
1 for (Node p = start; p != null; p = p.next)
2     System.out.println(p.data);
```

- 루프를 이용한 배열의 프린팅

LISTING 6: Using a for Loop to Print an Array

```
1 for (int i=0; i < n; i++)
2     System.out.println(a[i]);
```

Node 클래스의 테스팅

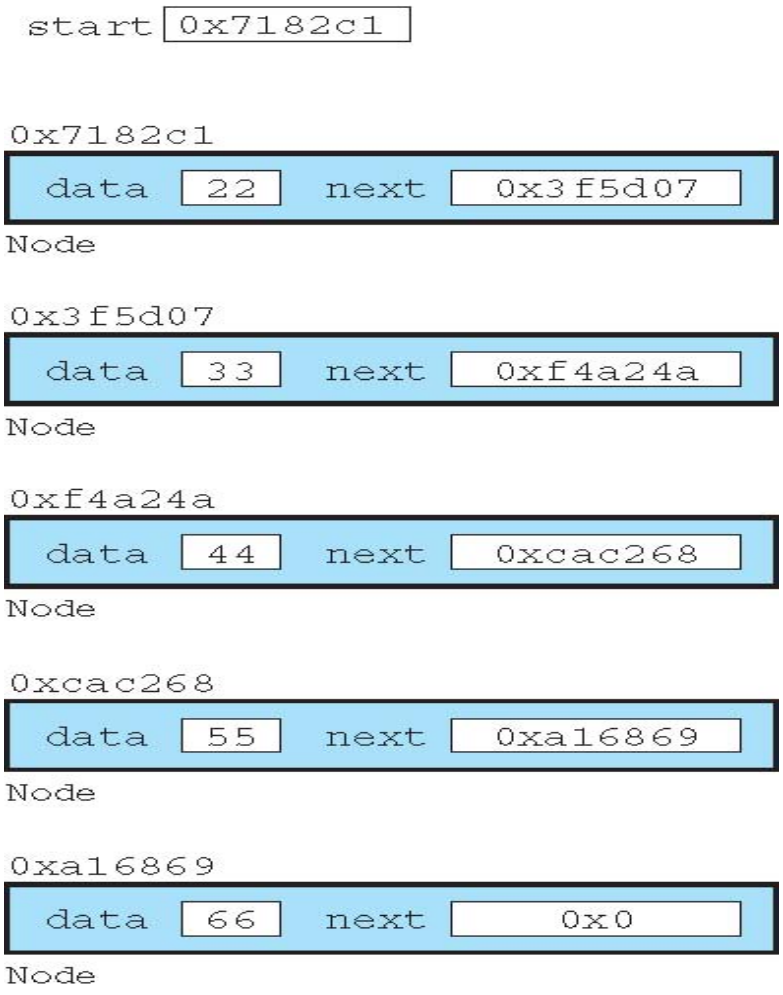
- **LISTING 7: Testing the Node Class**

```
1 public class TestNode {
2       public static void main(String[] args) {
3         Node start = new Node(22);
4         Node p=start;
5         for (int i = 1; i < 5; i++)
6               p = p.next = new Node(22+11*i);
7         for (p = start; p! = null; p = p.next)
8               System.out.println(p.data);
9         for (p = start; p != null; p = p.next)
10              System.out.println(p);
11    }
12 }
14 class Node {  // See Listing 1  }
```

출력 결과

- The output is

22
33
44
55
66
Node@7182c1
Node@3f5d07
Node@f4a24a
Node@cac268
Node@a16869



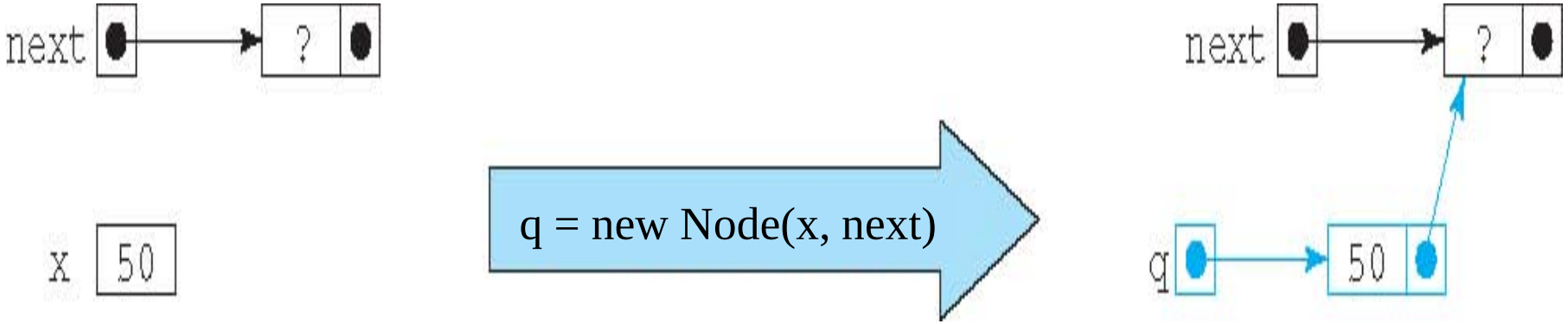
2. 연결 리스트에 대한 원소 삽입

- 삽입 과정의 단순화를 위해 2-인자 노드 생성자 추가
  - 노드의 생성과 삽입을 한꺼번에 수행할 수 있도록 해줌

- **LISTING 8: A Node Class with Two Constructors**

```
1 class Node {
2       int data;
3       Node next;

5       Node(int data) {
6           this.data = data;
7       }

9       Node(int data, Node next) {
10          this.data = data;
11          this.next = next;
12      }
13 }
```

2-인자 Node 생성자의 호출

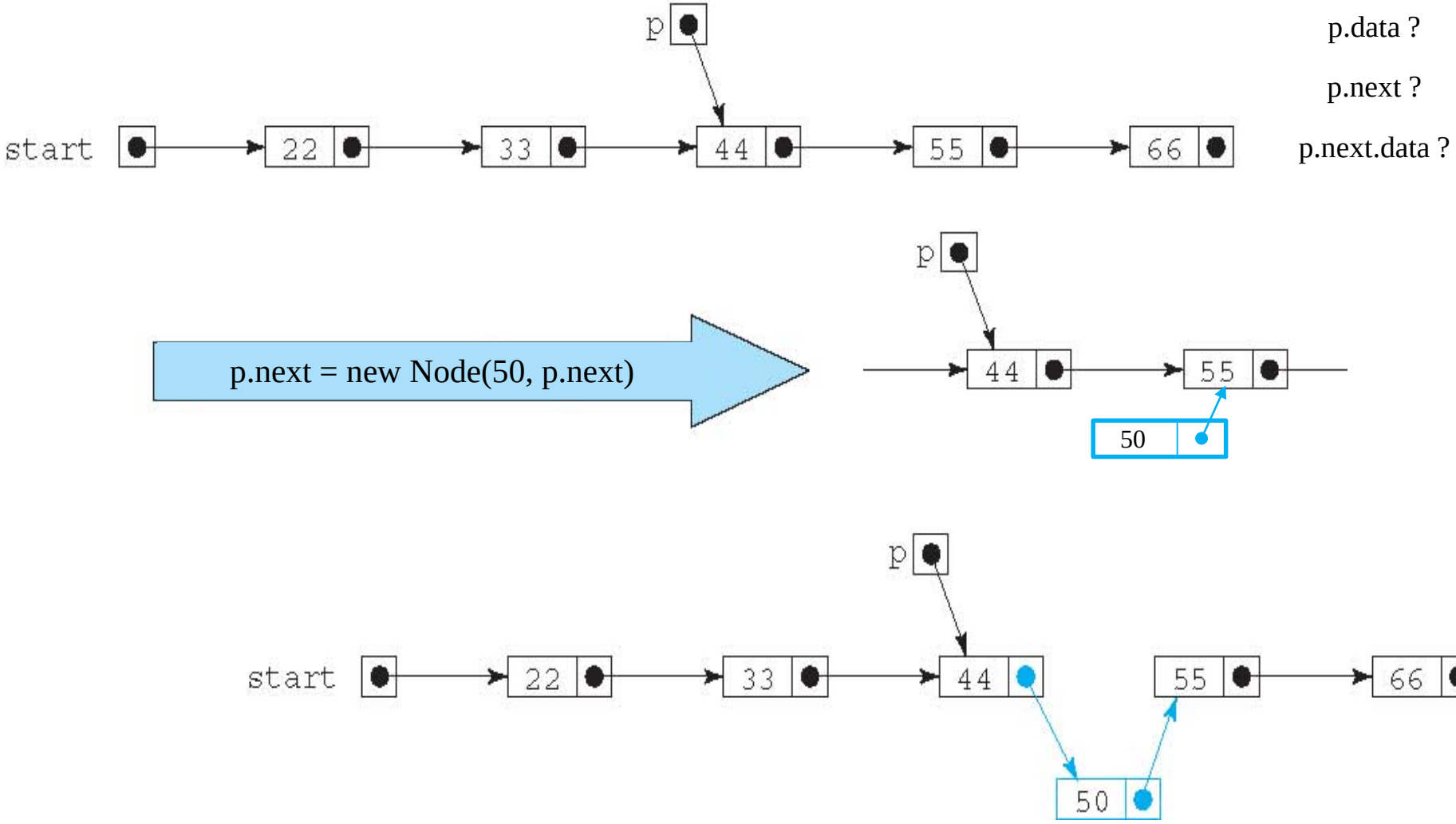


비공백 정렬 연결 리스트에 대한 삽입

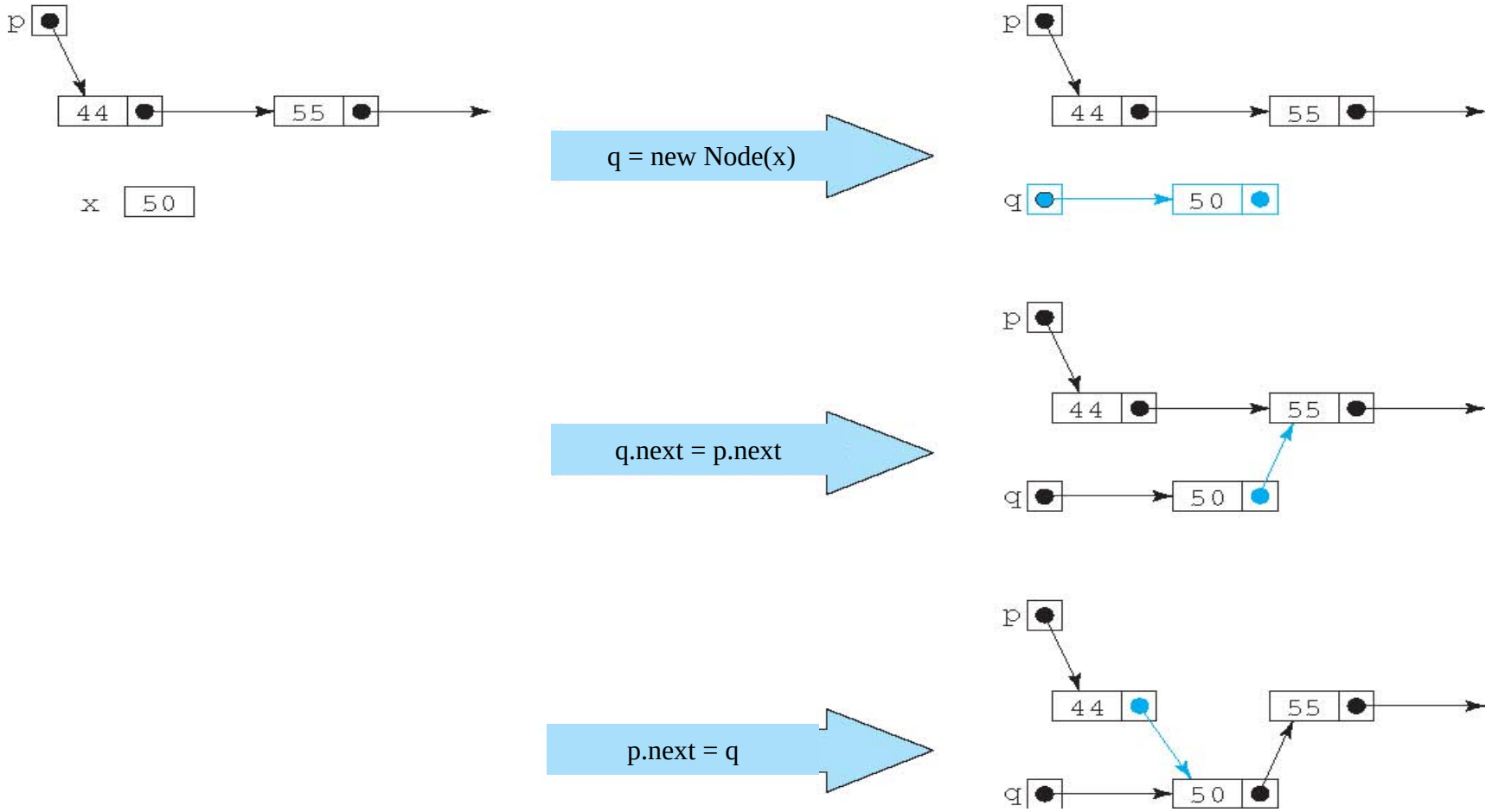
- 삽입
  - (1) 새로운 노드 앞에 놓일 리스트 노드 p 발견
  - (2) 새로운 노드를 생성해 부착
- **LISTING 9: Inserting into a Nonempty Sorted Linked List**

```
1 void insert(Node start, int x) {
2     // PRECONDITIONS: the list is in ascending order, and x > start.data;
3     // POSTCONDITIONS: the list is in ascending order, and it contains x;
4     Node p = start;
5     while (p.next != null) {
6     if (p.next.data > x) break;
7          p = p.next;
8      }
9      p.next = new Node(x,p.next);
10 }
```

비공백 정렬 연결 리스트에 대한 삽입

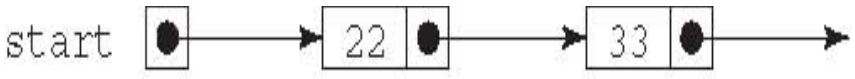


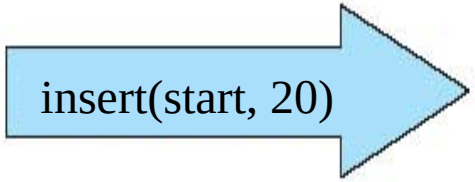
3단계에 걸친 새 노드의 삽입



3. 리스트의 앞에 삽입

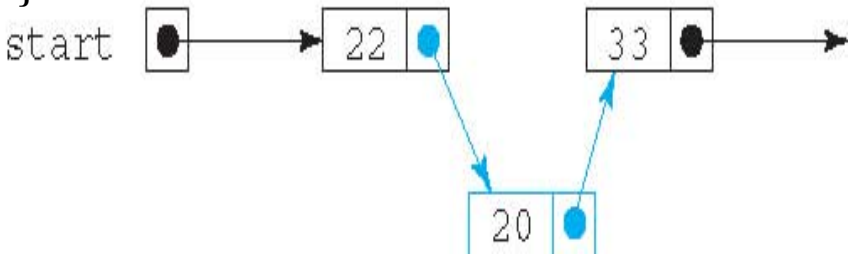
- LISTING 9의 insert() 메소드는 x가 리스트의 첫 번째 원소(start.data)보다 크다는 추가적인 선조건을 포함
  - 첫 번째 노드로 삽입시 새로운 노드 앞에 나올 노드가 결여되어 있기 때문
  - 첫 번째 노드로 삽입시 잘못된 위치에 삽입됨





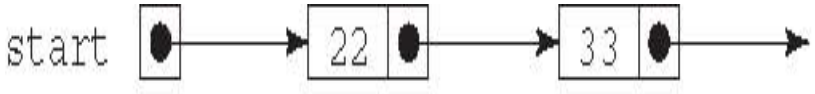
- 20을 틀리게 삽입

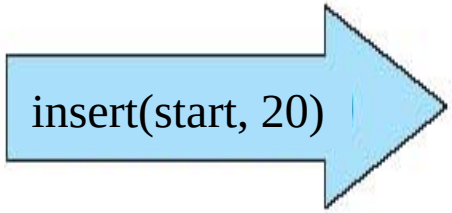
```
Listing 9
....
Node p = start;
while (p.next != null) {
    if (p.next.data > x) break;
    p = p.next;
}
```

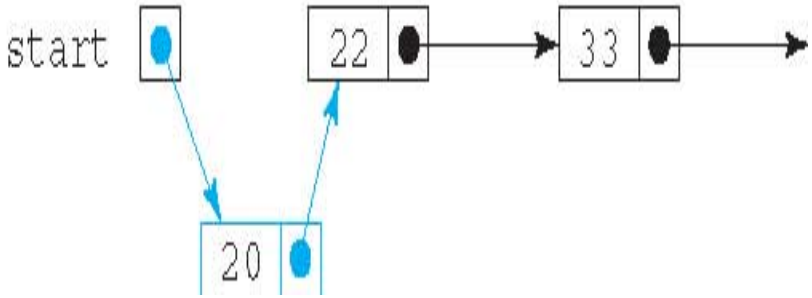


3. 리스트의 앞에 삽입

- 해결 방법
  - 방법1: 연결 리스트의 구조를 변경하여 첫 번째 실제 데이터 노드 앞에 "빈(dummy)" 헤드 노드를 유지
  - 방법2: LISTING 9의 insert() 메소드를 수정하여 이런 특별한 경우를 별도로 처리하도록 해주는 것 (LISTING 10)







- 20을 올바르게 삽입

연결 리스트 삽입

LISTING 10: Linked List Insertion

```
1 Node insert(Node start, int x) {
2   // precondition: the list is in ascending order;
3   // postconditions: the list is in ascending order, & it contains x;
4       if (start == null || start.data > x) {
5           start = new Node(x, start);
6           return start;
7       }
8       Node p=start;
9       while (p.next != null) {
10          if (p.next.data > x) break;
11          p = p.next;
12      }
13      p.next = new Node(x, p.next);
14      return start;
15 }
```

4. 정렬된 연결 리스트에서의 삭제

- delete() 메소드
  - (1) 원소 발견
  - (2) 삭제

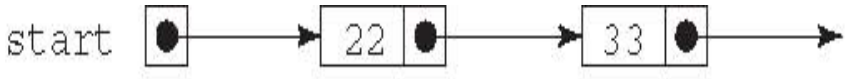
연결 리스트 삭제

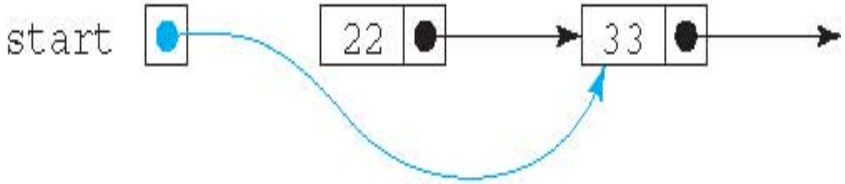
- **LISTING 11: Linked List Deletion**

```
1 Node delete(Node start, int x) {
2       // precondition: the list is in ascending order;
3       // postconditions: the list is in ascending order, and if it did
4       // contains x, then the first occurrence of x has been deleted;
5       if (start == null || start.data > x) // x is not in the list
6           return start;
7       if (start.data==x) // x is the first element in the list
8           return start.next;
9       for (Node p = start; p.next != null; p = p.next) {
10          if (p.next.data > x) break; // x is not in the list
11          if (p.next.data == x) { // x is in the p.next node
12              p.next = p.next.next; // delete it
13              break;
14          }
15      }
16      return start;
17 }
```

정렬된 연결 리스트에서 원소의 삭제

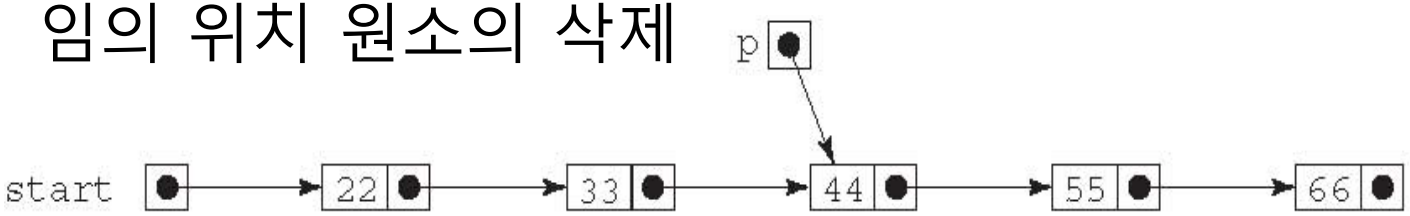
첫 번째 원소의 삭제

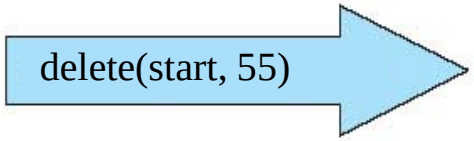




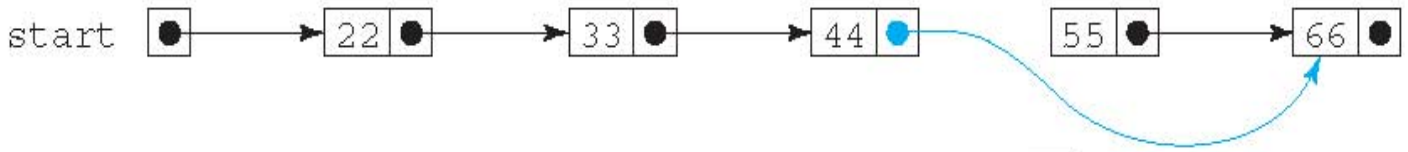
정렬된 연결 리스트에서 원소의 삭제

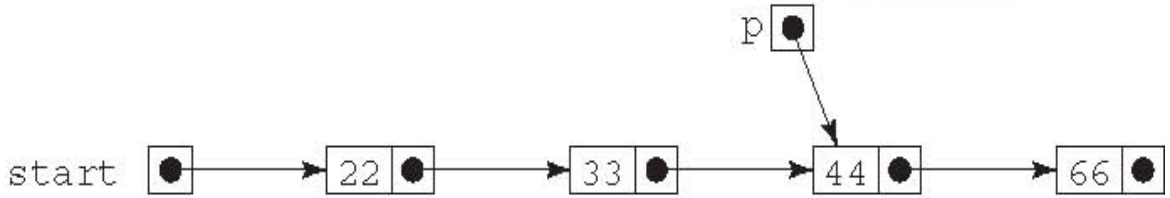
임의 위치 원소의 삭제





p.next = p.next.next;



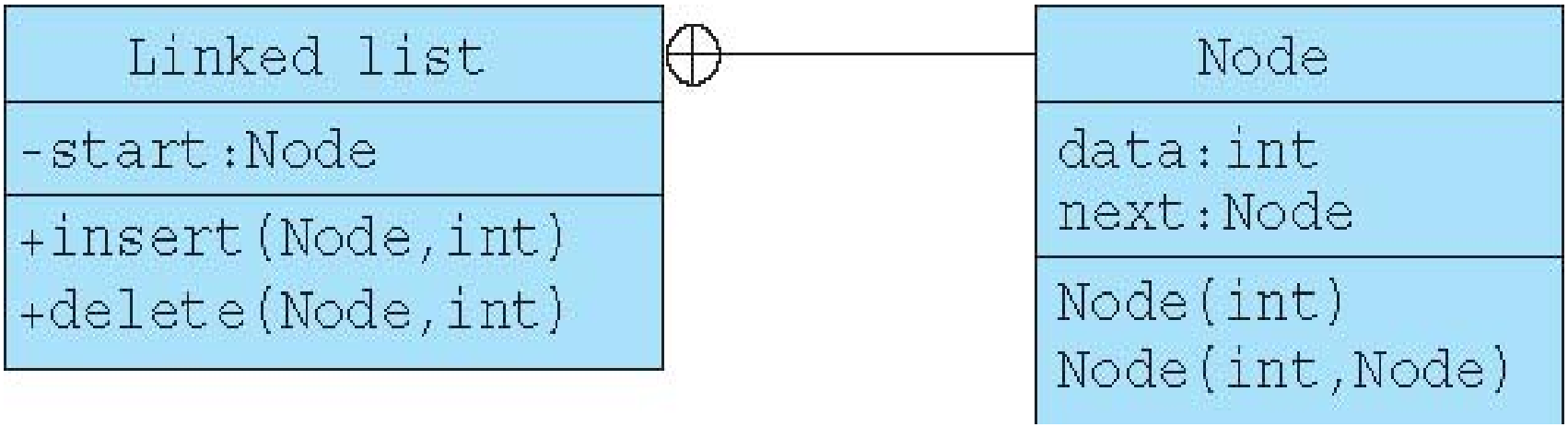


5. 중첩 클래스

- Java의 클래스 멤버
  - 필드, 생성자, 메소드, 인터페이스, 또다른 클래스 등

- 중첩 클래스(nested class)
  - 다른 클래스의 멤버인 클래스
  - 클래스 Y가 사용될 유일한 장소가 다른 클래스 X의 내부인 경우 클래스 Y는 클래스 X 안에 중첩되어야 함

LinkedList 노드 안에 중첩된 Node 클래스



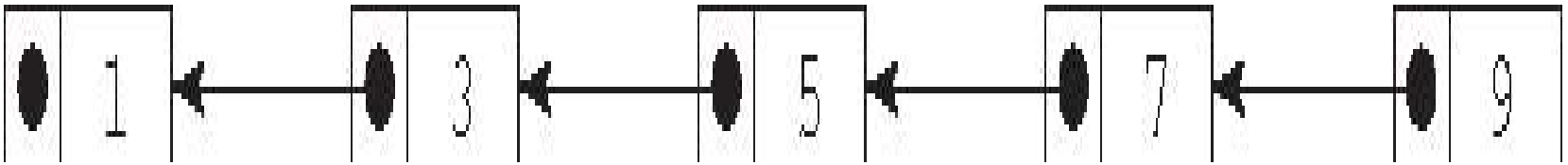
LinkedList 클래스 안에 중첩된 Node 클래스

- **LISTING 12: Nesting the Node Class within a LinkedList Class**

```
1 public class LinkedList {
2       private Node start;

4       public void insert(int x) {
5           // See Listing 10
6       }

8       public void delete(int x) {
9           // See Listing 11
10      }

12      private static class Node {
13          // See Listing 8
14      }
15 }
```

6. 사례 연구: 임의의 긴 정수

- 19자리 이상의 정수가 필요할 경우 임의의 길이의 정수를 허용하는 java.math.BigInteger 클래스를 사용
- 연결 리스트를 이용하여 이러한 정수 객체 구축 가능
- 정수 13,579를 표현하는 연결 리스트



BigInt 클래스의 private 멤버들

- **LISTING 13: The private Members of a BigInt Class**

```
1 public class BigInt {
2       private Node start;
3
4       private static class Node {
5           int digit;
6           Node next;
7           Node(int digit) { this.digit = digit; }
8       }
9 }
```

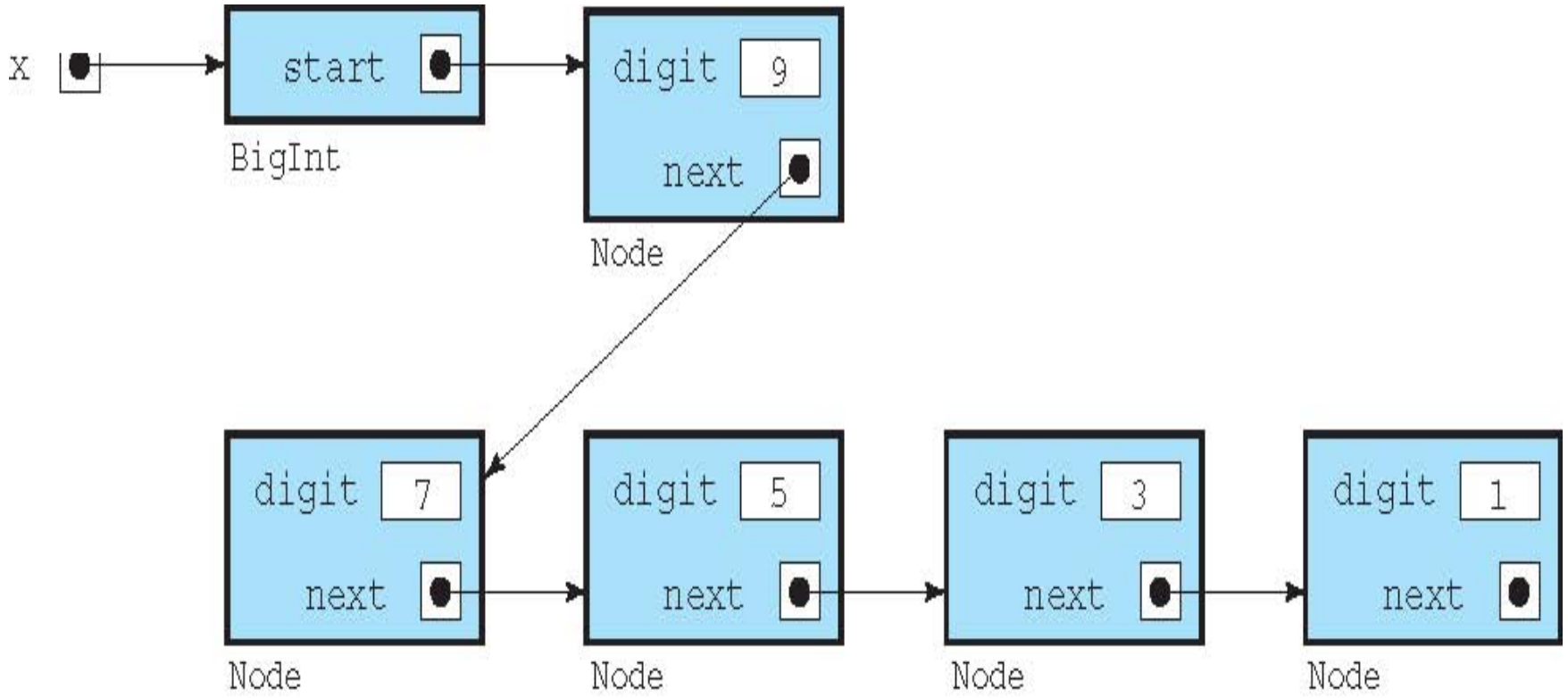
주어진 int에 대한 BigInt 클래스 생성자

- 일반 정수로부터 BigInt 객체를 만드는 생성자

- **LISTING 14: The BigInt Class Constructor for a Given int**

```
1 public BigInt(int n) {     // 정수 13579와 같은 경우
2       if (n<0) throw new IllegalArgumentException(n+"<0");
3       start = new Node(n%10);
4       Node p=start;
5       n /= 10;
6       while (n>0) {
7           p = p.next = new Node(n%10);
8           n /= 10;
9       }
10 }
```

정수 13,579를 표현하는 BigInt



주어진 String에 대한 BigInt 클래스 생성자

- **LISTING 15: The BigInt Class Constructor for a Given String**

```
1 public BigInt(String s) {   // 문자열 "13579"와 같은 경우
2       if (s.length() == 0)
3           throw new IllegalArgumentException("empty string");
4       start = new Node(digit(s, s.length()-1));
5       Node p=start;
6       for (int i = s.length()-2; i >= 0; i--)
7           p = p.next = new Node(digit(s, i));
8 }

10 private int digit(String s, int i) {
11     String ss = s.substring(i, i+1);
12     return Integer.parseInt(ss);
13 }
```

BigInt 클래스를 위한 toString() 메소드

- BigInt 객체를 정수로 프린트하는 toString() 메소드
  - 해당 자리들을 누적하기 위해 StringBuffer 객체 사용

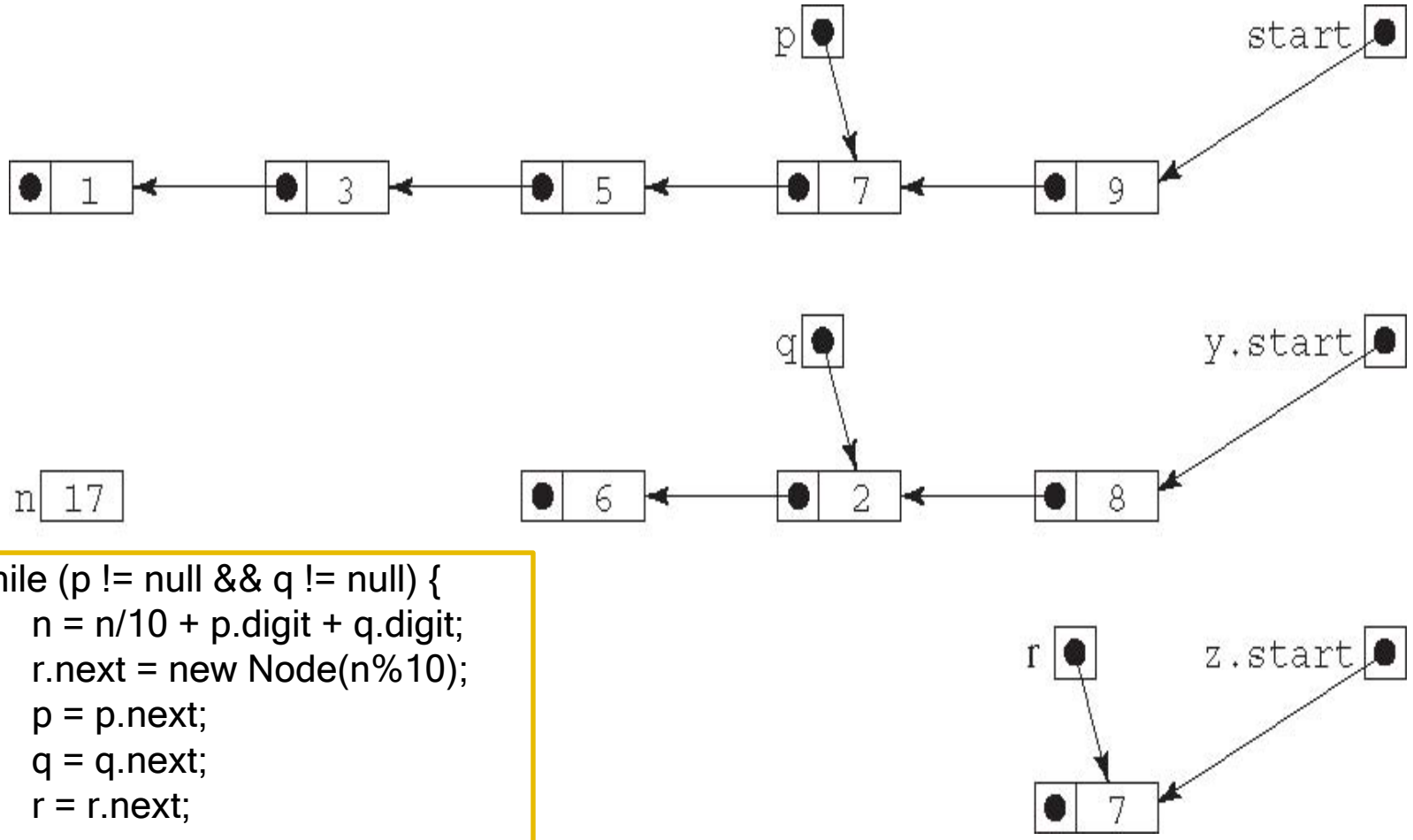
- **LISTING 16: A toString() Method for the BigInt Class**

```
1 public String toString() {
2       StringBuffer buf = new
            StringBuffer(Integer.toString(start.digit));
3       Node p = start.next;
4       while (p != null) {
5           buf.insert(0,Integer.toString(p.digit));
6           p = p.next;
7       }
8       return buf.toString();
9 }
```

BigInt 객체를 더하는 메소드

```
1 public BigInt plus(BigInt y) {
2       Node p = start, q = y.start;
3       int n = p.digit + q.digit;
4       BigInt z = new BigInt(n%10);
5       Node r=z.start;   p = p.next;   q = q.next;
8       while (p != null && q != null) {
9           n = n/10 + p.digit + q.digit;   r.next = new Node(n%10);
11          p = p.next;   q = q.next;   r = r.next;
14      }
15      while (p != null) {
16          n = n/10 + p.digit;   r.next = new Node(n%10);
18          p = p.next;   r = r.next;
20      }
21      while (q != null) {
22          n = n/10 + q.digit;   r.next = new Node(n%10);
24          q = q.next;   r = r.next;
26      }
27      if (n > 9) r.next = new Node(n/10);
28      return z;
29 }
```

BigInt 628과 BigInt 13,579의 덧셈




```
while (p != null && q != null) {
        n = n/10 + p.digit + q.digit;
        r.next = new Node(n%10);
        p = p.next;
        q = q.next;
        r = r.next;
}
```

Q & A