

4. 연결 구조

개요

- 신속한 삽입과 삭제를 허용하는 순서 리스트 유지에 적합

4.1 순서 배열의 유지

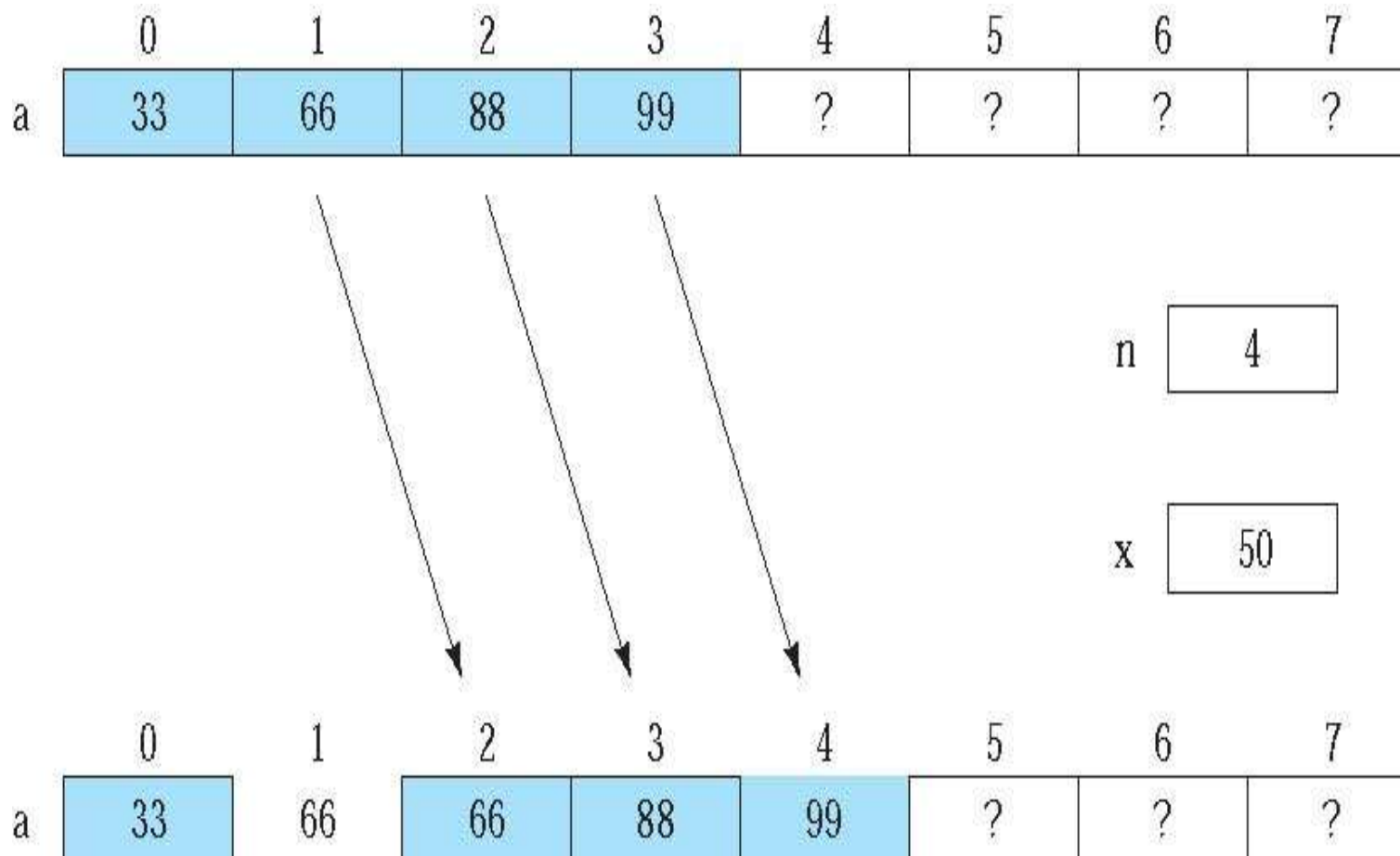
- 정렬된 배열에 새 원소 삽입
 - 배열 구현의 경우 새 원소 보다 큰 원소들을 모두 이동시켜야 함

순서 배열로의 삽입

- LISTING 4.1: Listing 4.1 Inserting into an Ordered Array

```
1 int[] insert(int[] a, int n, int x) {
2     // preconditions: a[0] <= ... <= a[n-1], and n < a.length;
3     // postconditions: a[0] <= ... <= a[n], and x is among them;
4     int i = 0;
5     // find the first index i for which a[i] > x
5     while (i < n && a[i] <= x)
6         ++i;
7     // shift {a[i],...,a[n-1]} into {a[i+1],...,a[n]}
8     System.arraycopy(a, i, a, i+1, n-i);
9     // insert x into a[i]
10    a[i] = x;
11 }
```

새 원소에 대한 공간 확보



x를 올바른 위치에 복사

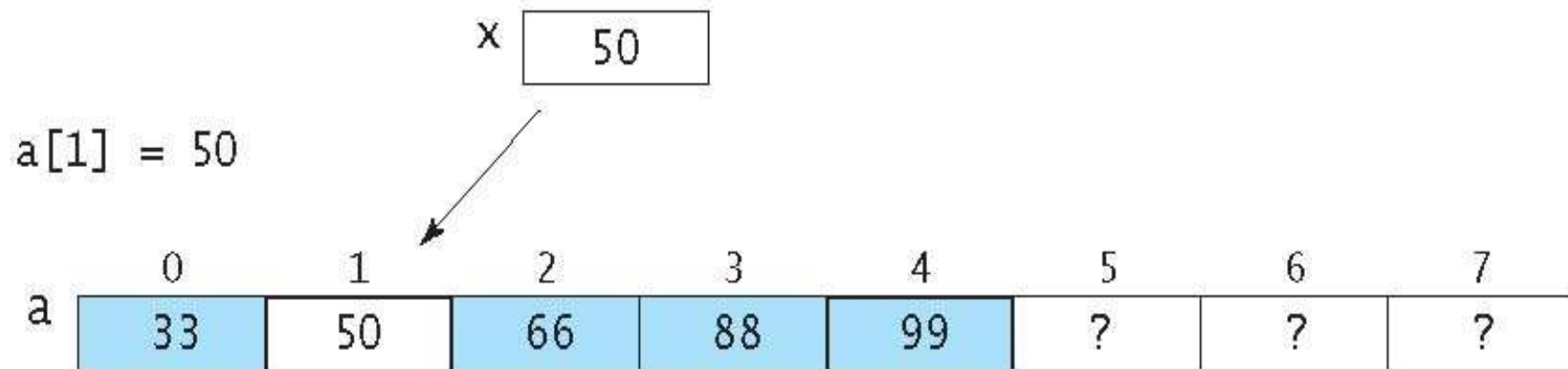
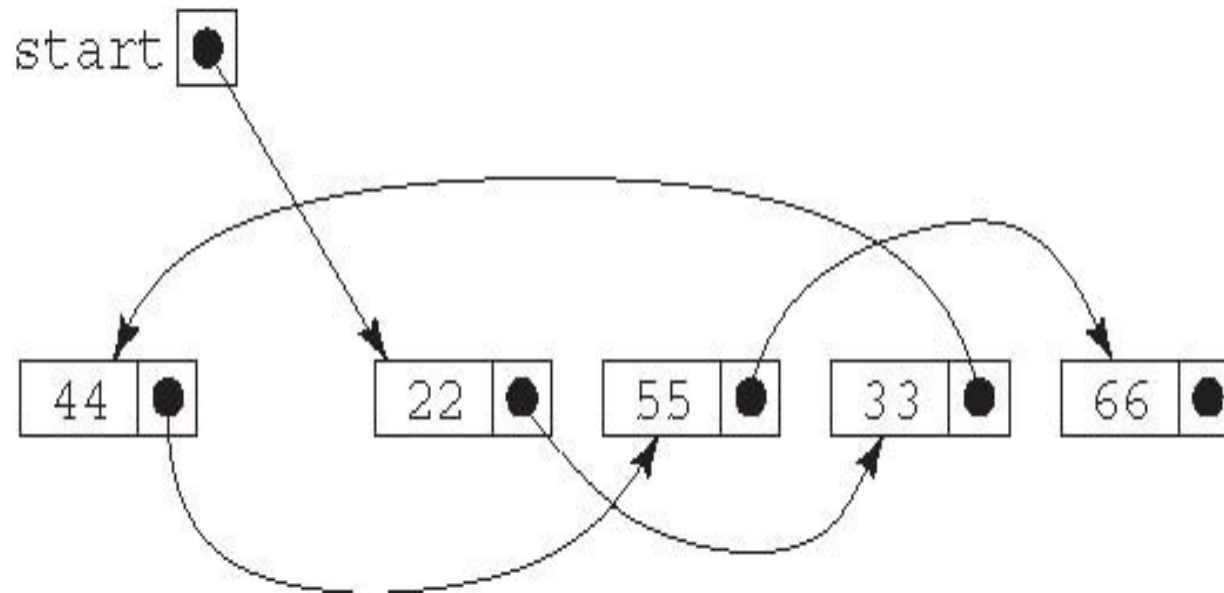


FIGURE 4.2 Copying x into its correct position

4.3 연결 노드

- [개선안] 원소에 그 참조를 위한 객체의 사용
- 객체-지향적인 관점에서는 이를 Node 객체의 시퀀스로 생각할 수 있음
 - 배열 `a[]` 대신에 단일 `start` 참조만을 유지

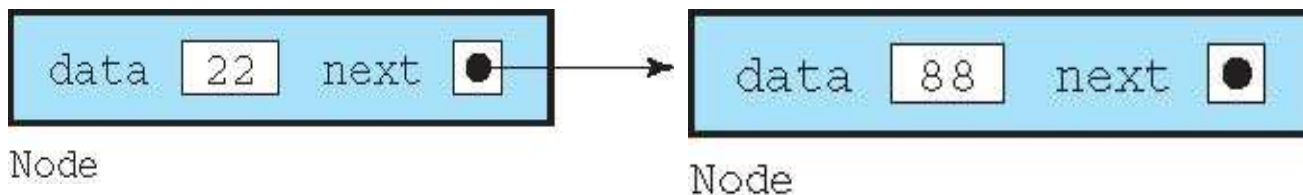


Node 클래스

- Node 클래스는 자기 참조 형태로 정의

```
class Node {  
    private int data;  
    private Node next;  
  
    public Node(int data) {  
        this.data = data;  
    }  
}
```

- 전형적인 Node 객체

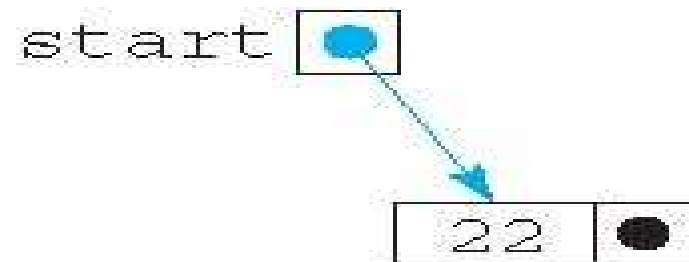


연결 리스트의 생성

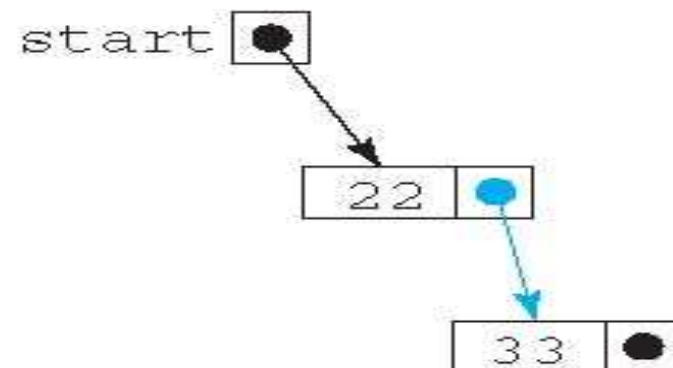
- 5-원소 리스트의 생성
 - LISTING 4.4: Constructing a Linked List

```
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);
```

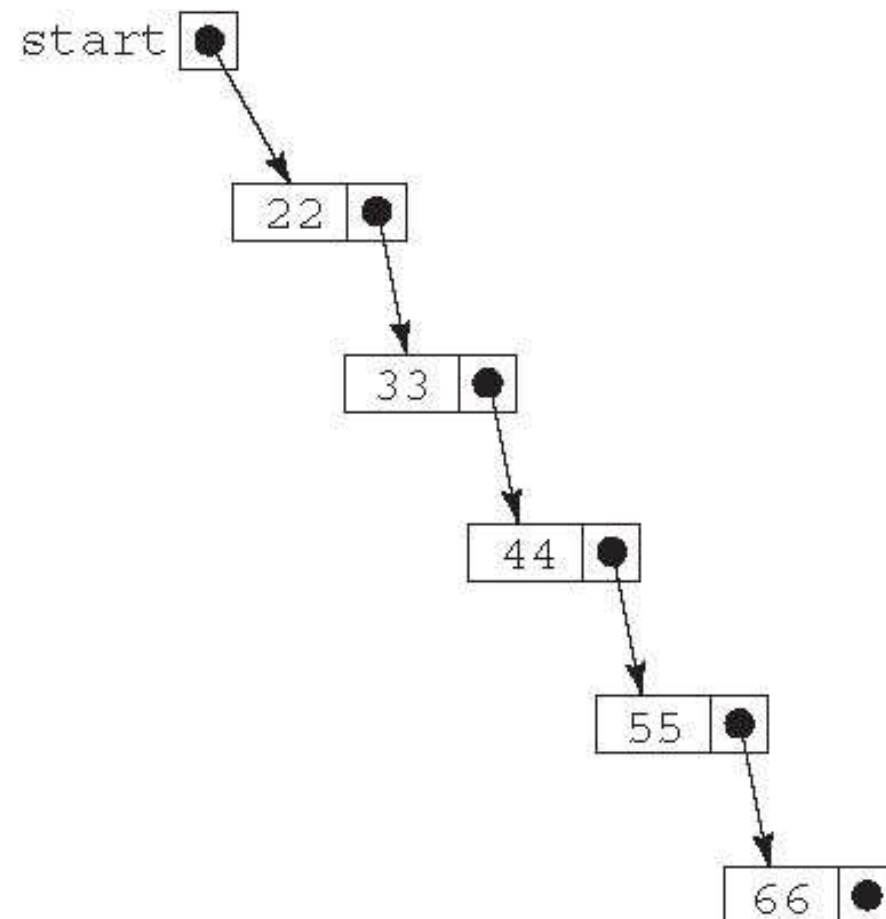
- start의 초기화



- 한 노드의 추가



5-노드 리스트



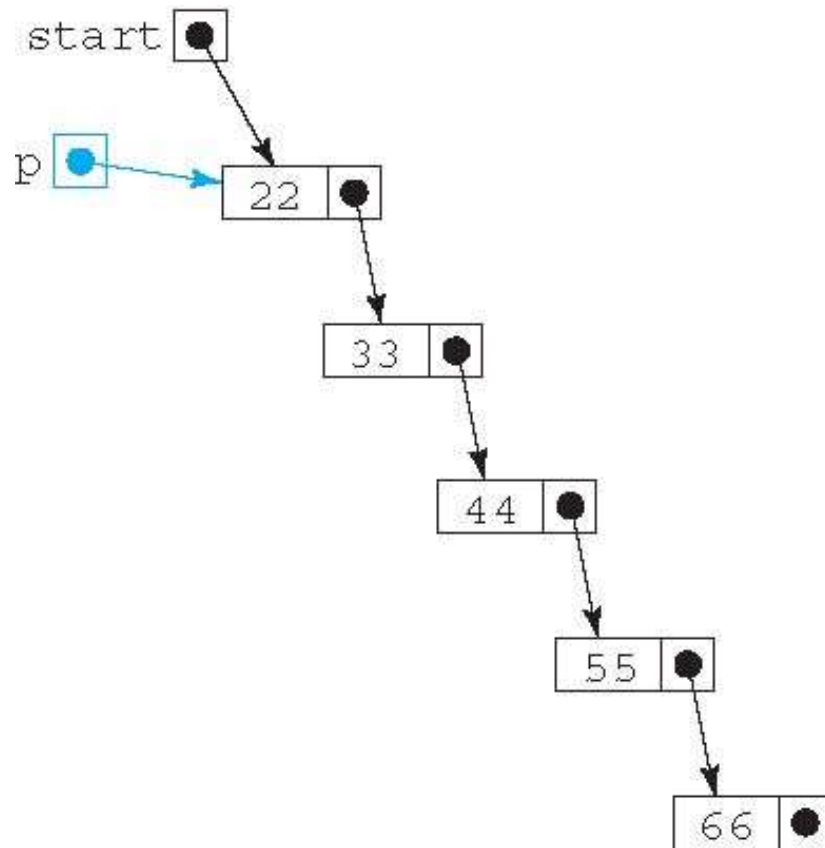
연결 리스트의 생성

- 리스트를 통해 “**걸어 나갈 수**” 있는 지역 참조 변수의 사용
 - 전통적으로 변수 p(포인터를 의미)를 사용
 - 변수 p는 Node 참조로 선언
 - 연결 리스트의 생성
- LISTING 4.5: 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);
```

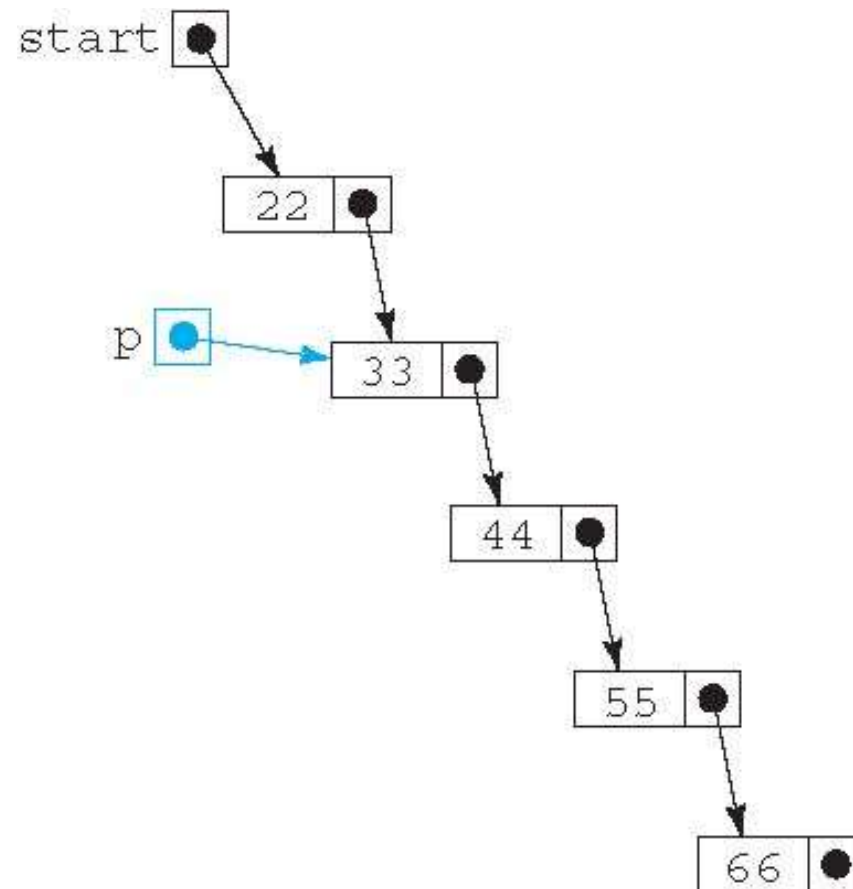
p를 start 노드로 초기화

Node p=start;



p를 두 번째 노드로 전진


`p = p.next;`



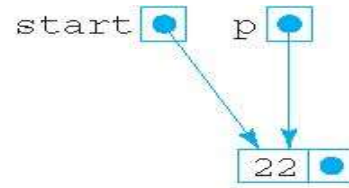
for 루프의 사용

- 루프를 이용한 동일한 연결 리스트의 생성

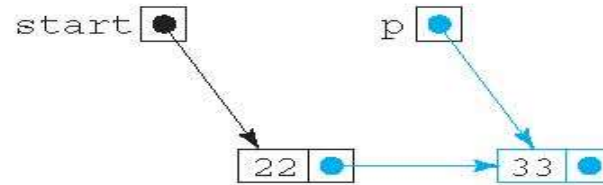
LISTING 4.6: Using a for Loop

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

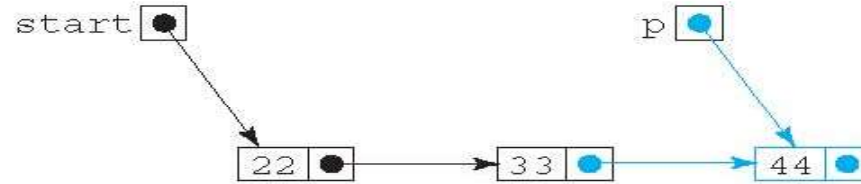
`p = p.next = new Node(22)`



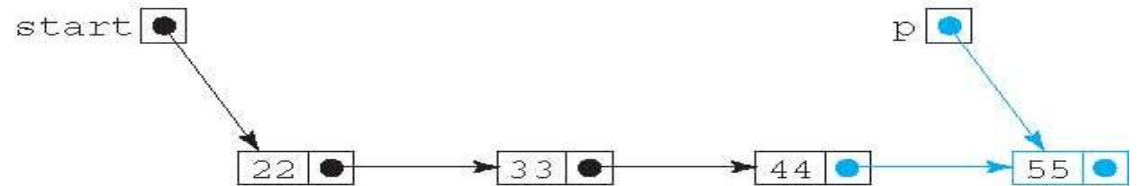
`p = p.next = new Node(33)`



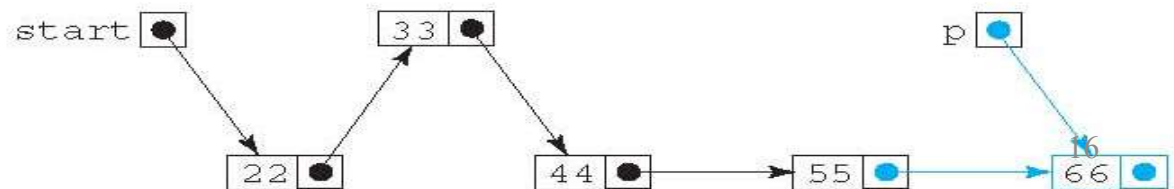
`p = p.next = new Node(44)`



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




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

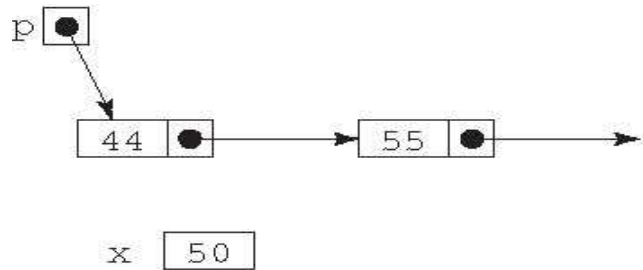


루프를 이용한 연결 리스트의 프린팅

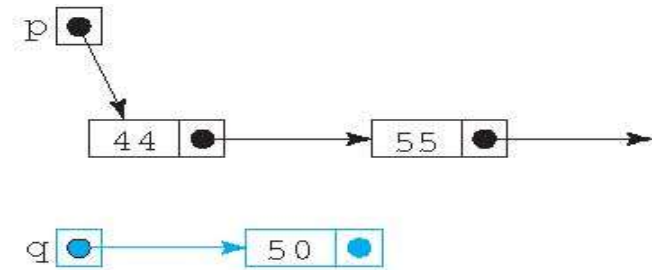
LISTING 4.7: Using a for Loop to Print a Linked List

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

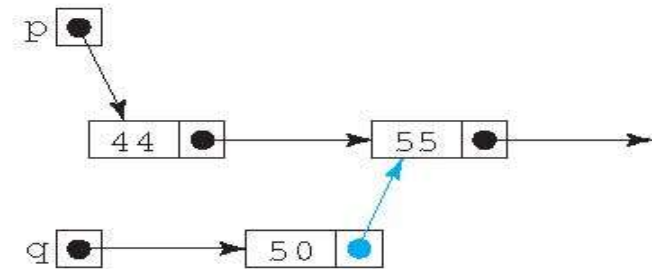
3 단계에 걸친 새 노드의 삽입



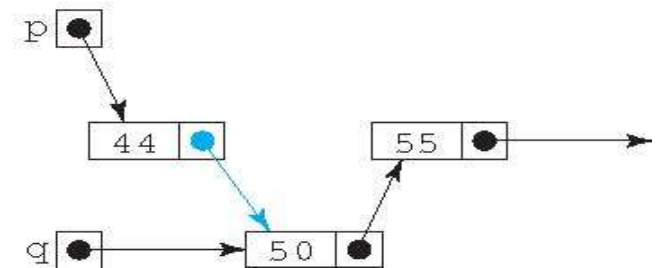
$q = \text{new Node}(x)$



$q.\text{next} = p.\text{next}$



$p.\text{next} = q$



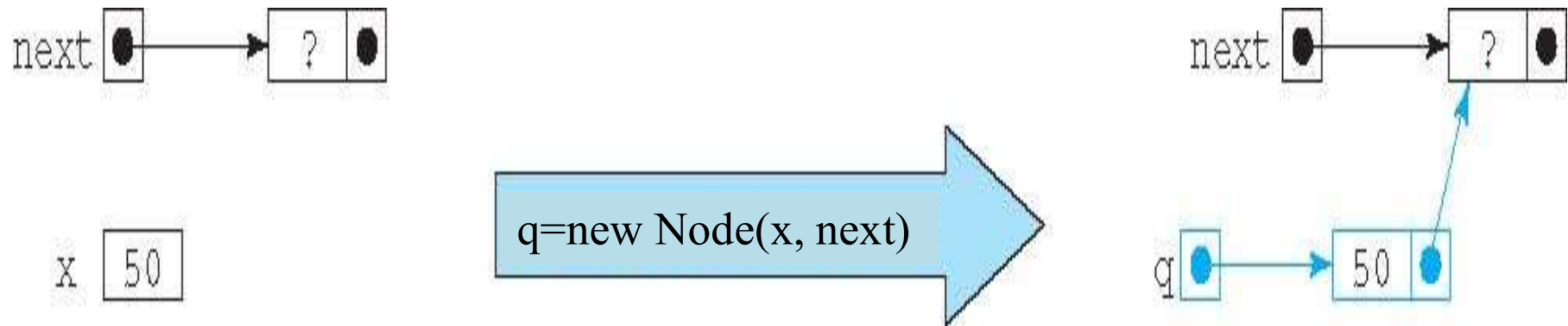
4.4 연결 리스트에 대한 원소 삽입

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

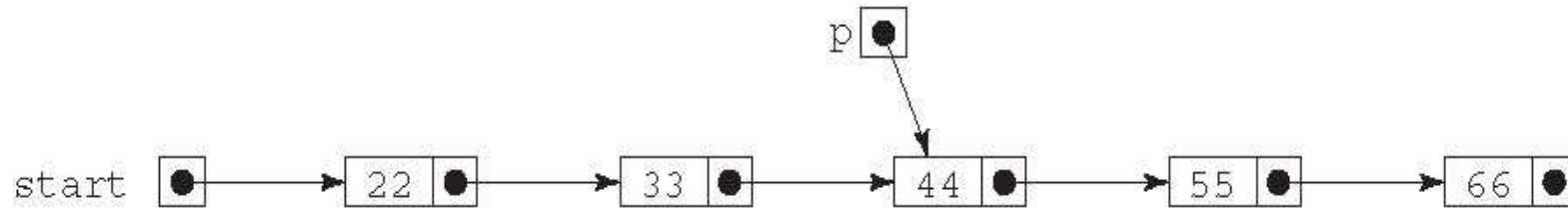
- LISTING 4.10: A Node Class with Two Constructors

```
1 class Node {  
2     int data;  
3     Node next;  
4  
5     Node(int data) {  
6         this.data = data;  
7     }  
8  
9     Node(int data, Node next) {  
10        this.data = data;  
11        this.next = next;  
12    }  
13 }
```

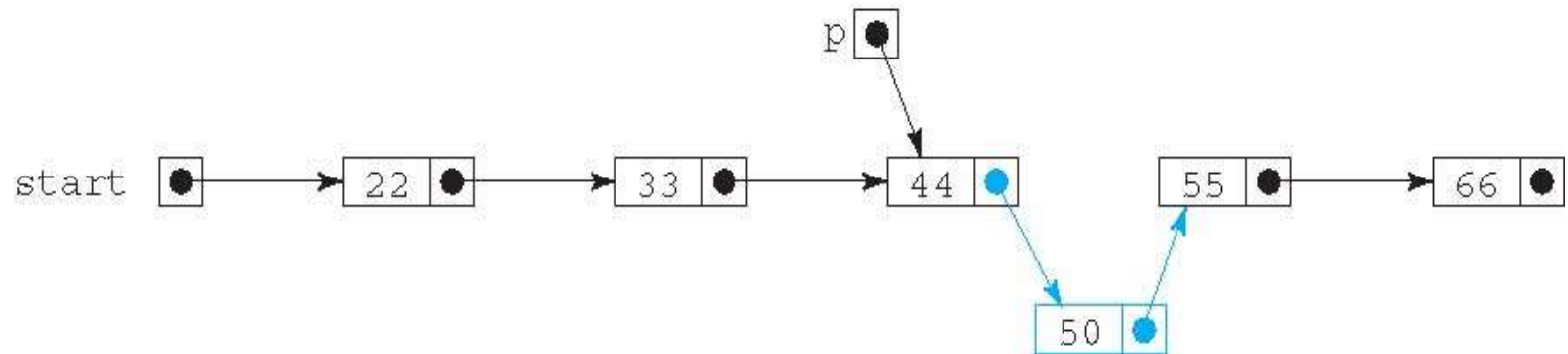
2-인자 Node 생성자의 호출



비공백 정렬 연결 리스트에 대한 삽입



`p.next=new Node(50, p.next)`



비공백의 정렬된 정수 연결 리스트에 대한 삽입

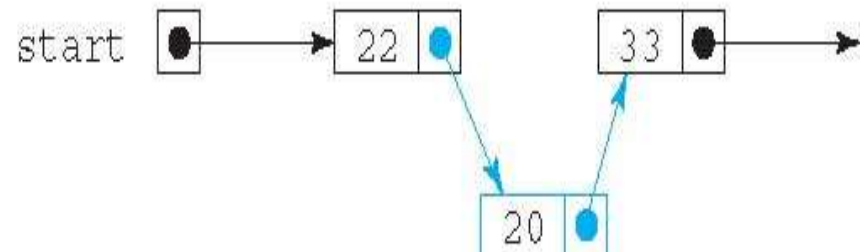
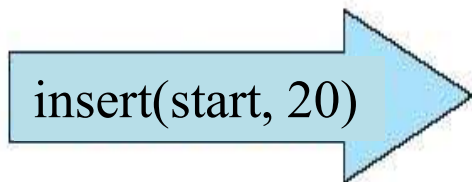
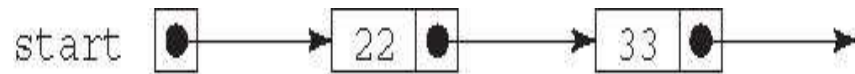
- (1) 새로운 노드 앞에 놓일 리스트 노드 p 발견
- (2) 새로운 노드를 생성해 부착

- LISTING 4.11:

```
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 }
```

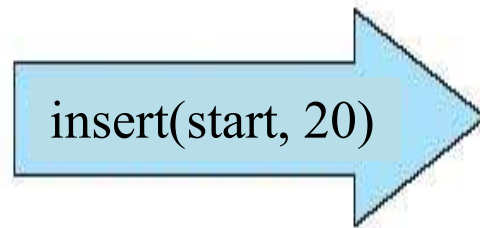
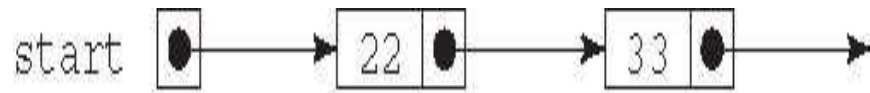
4.5 리스트의 앞에 삽입

- 리스칭 4.11의 insert() 메소드는 x가 리스트의 첫 번째 원소 (start.data) 보다 크다는 추가적인 선 조건을 포함
 - 첫 번째 노드로 삽입시 새로운 노드 앞에 나올 노드가 결여되어 있기 때문
 - 첫 번째 노드로 삽입시 잘못된 위치에 삽입됨

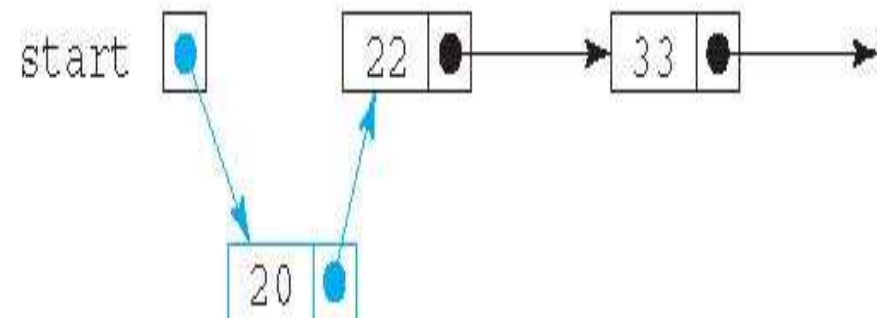


- 해결 방법

- 방법1: 연결 리스트의 구조를 변경하여 첫 번째 실제 데이터 노드 앞에 "빈(dummy)" 헤드 노드를 유지
- 방법2: 리스팅 4.11의 insert() 메소드를 수정하여 이런 특별한 경우를 별도로 처리하도록 해주는 것 (리스팅 4.12)



- 20을 올바르게 삽입



- LISTING 4.12:

```
1 Node insert(Node start, int x) {  
2   // precondition: the list is in ascending order;  
3   // postconditions: the list is in ascending order, and 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 }
```

4.6 정렬된 연결 리스트에서의 삭제

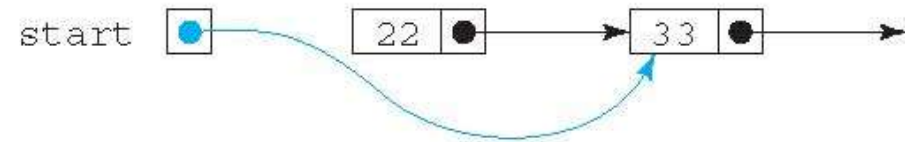
- delete() 메소드
 - (1) 원소 발견
 - (2) 삭제

정렬된 연결 리스트에서 첫 번째 원소의 삭제

첫 번째 원소의 삭제



delete (start, 22)

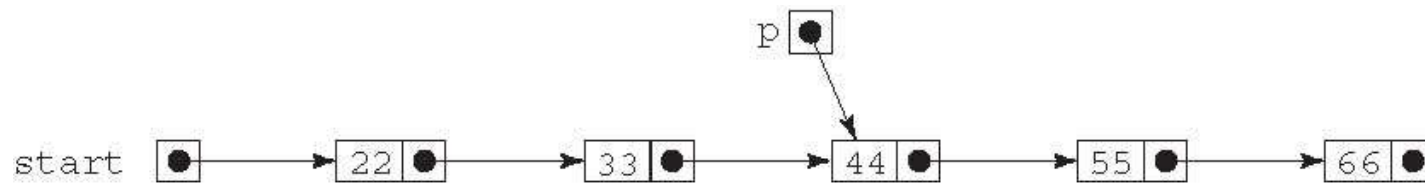


garbage collected

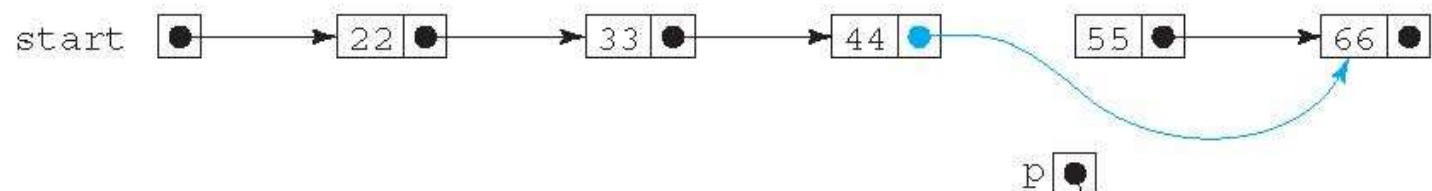


정렬된 연결 리스트에서 원소의 삭제

임의 위치 원소의 삭제



delete (start, 55)



garbage collected



- LISTING 4.13:

```
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     // contain 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 [redacted]
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 = [redacted] // delete it
13             break;
14         }
15     }
16     return start;
17 }
```

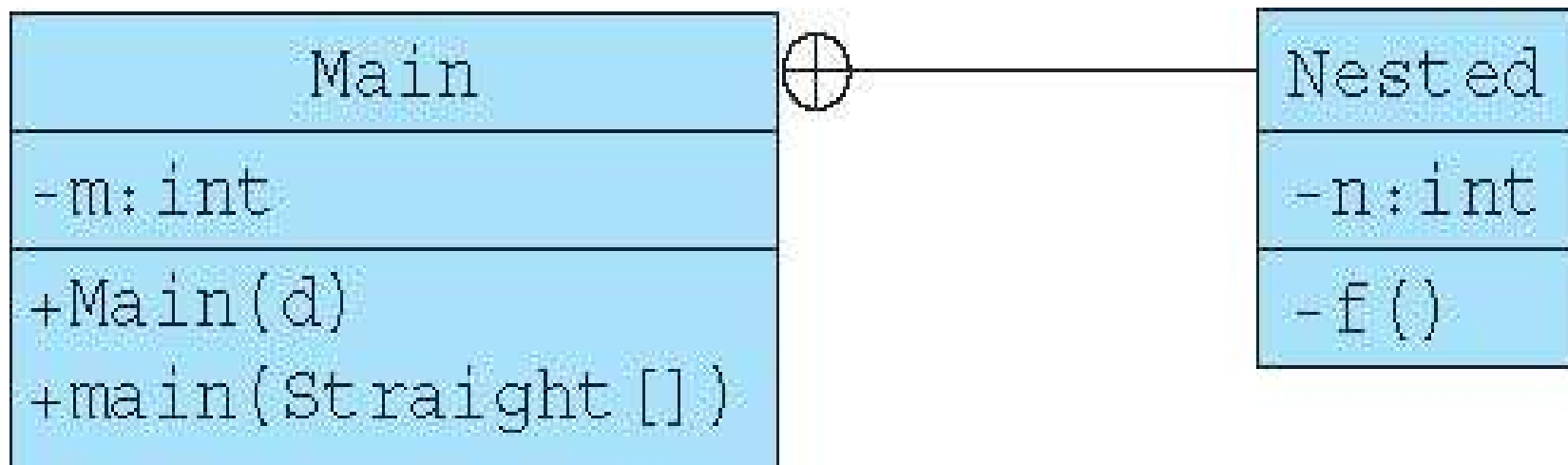
4.7 중첩 클래스

- Java의 클래스 멤버
 - 필드, 생성자, 메소드, 인터페이스, 또 다른 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

중첩 클래스로부터의 접근 가능성(LISTING 4.14)

```
1 public class Main {
2     private int m = 22;
4     public Main( ) {
5         Nested nested = new Nested();
6         System.out.println("Outside of Nested; nested.n = " + nested.n);
7         nested.f( );
8     }
10    public static void main(String[] args) {
11        new Main( );
12    }
14    private class Nested {
15        private int n = 44;
17        private void f( ) {
18            System.out.println("Inside of Nested; m = " + m);
19        }
20    }
21 }
```

중첩 클래스에 대한 UML 다이어그램

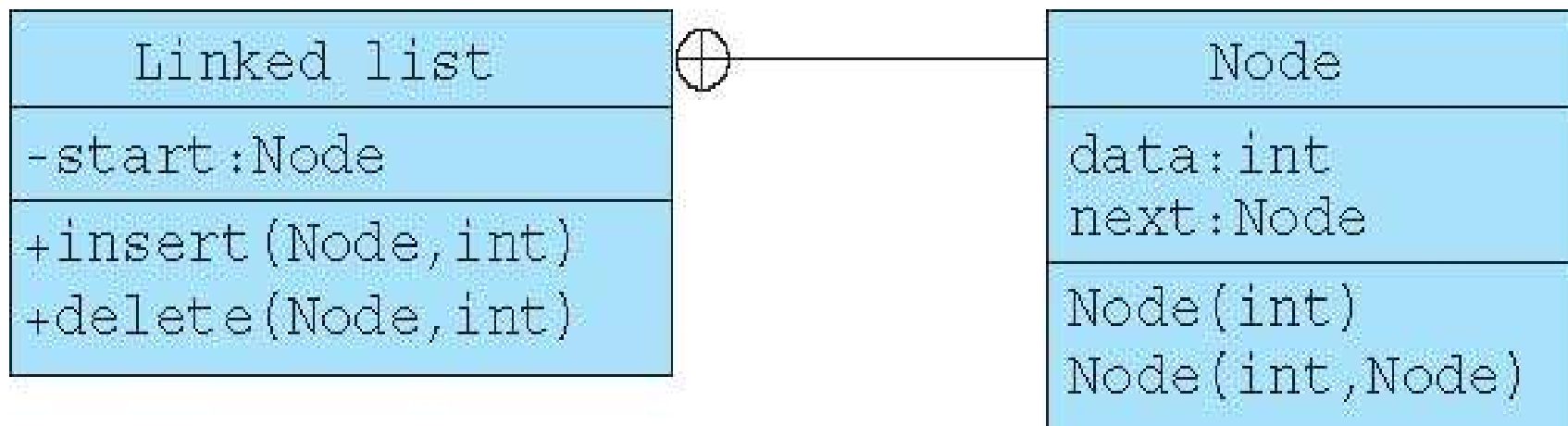


LinkedList 클래스 안에 중첩된 Node 클래스

- LISTING 4.15: Nesting the Node Class within a LinkedList Class

```
1 public class LinkedList {  
2     private Node start;  
3  
4     public void insert(int x) {  
5         // See Listing 4.12 on page 155  
6     }  
7  
8     public void delete(int x) {  
9         // See Listing 4.13 on page 156  
10    }  
11  
12    private static class Node {  
13        // See Listing 4.10 on page 150  
14    }  
15 }
```

LinkedList 노드 안에 중첩된 Node 클래스



4.8 사례 연구: 임의의 긴 정수

- 19 자리 이상의 정수가 필요할 경우 임의의 길이의 정수를 허용하는 `java.math.BigInteger` 클래스를 사용
- 연결 리스트를 이용하여 이러한 정수 객체 구축 가능

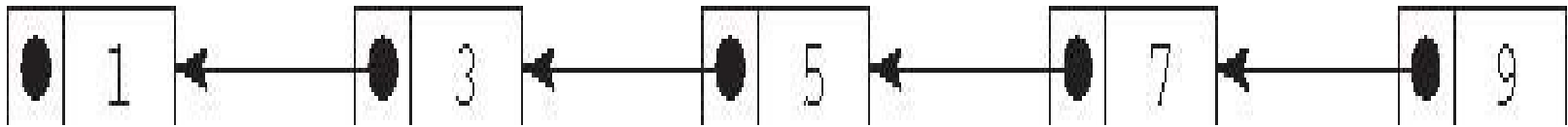


그림 4.27 정수 13,579를 표현하는 연결 리스트

BigInt 클래스의 private 멤버들

- LISTING 4.16: The private Members of a BigInt Class

```
public class BigInt {  
    private Node start;  
  
    private static class Node {  
        int digit;  
        Node next;  
        Node(int digit) { this.digit = digit; }  
    }  
    .....  
}
```

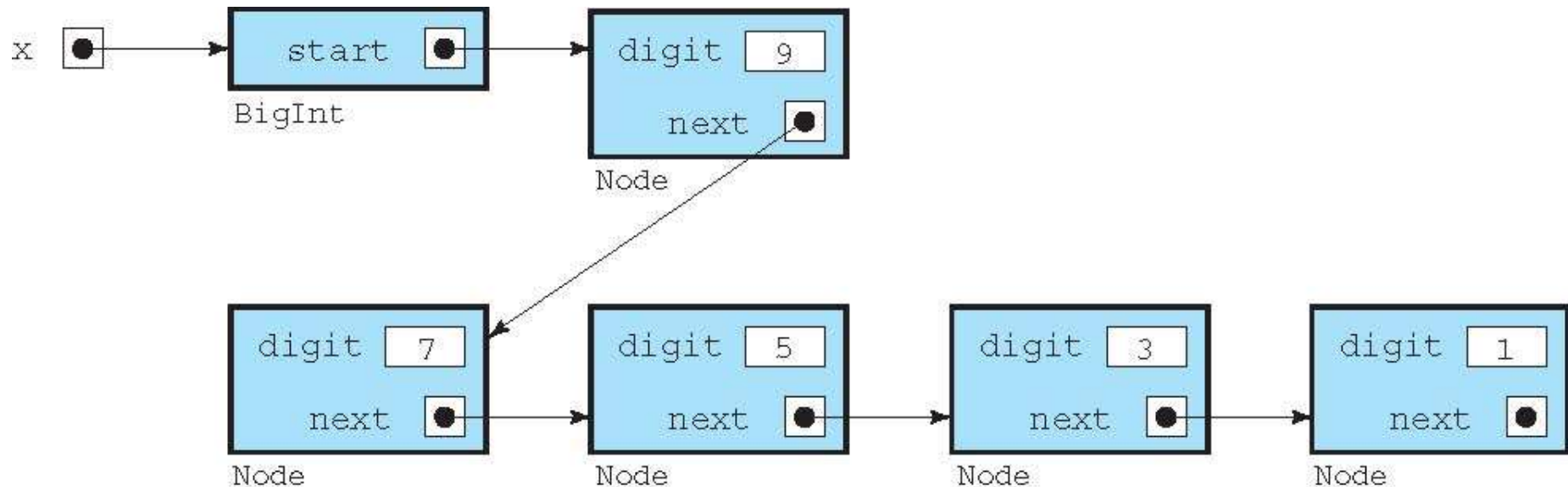
주어진 int에 대한 BigInt 클래스 생성자

(LISTING 4.17) 일반 정수로부터 BigInt 객체를 만드는 생성자

```
1 public BigInt(int n) {  
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

BigInt x = new BigInt(13579)



주어진 String에 대한 BigInt 클래스 생성자

- LISTING 4.18: The BigInt Class Constructor for a Given String

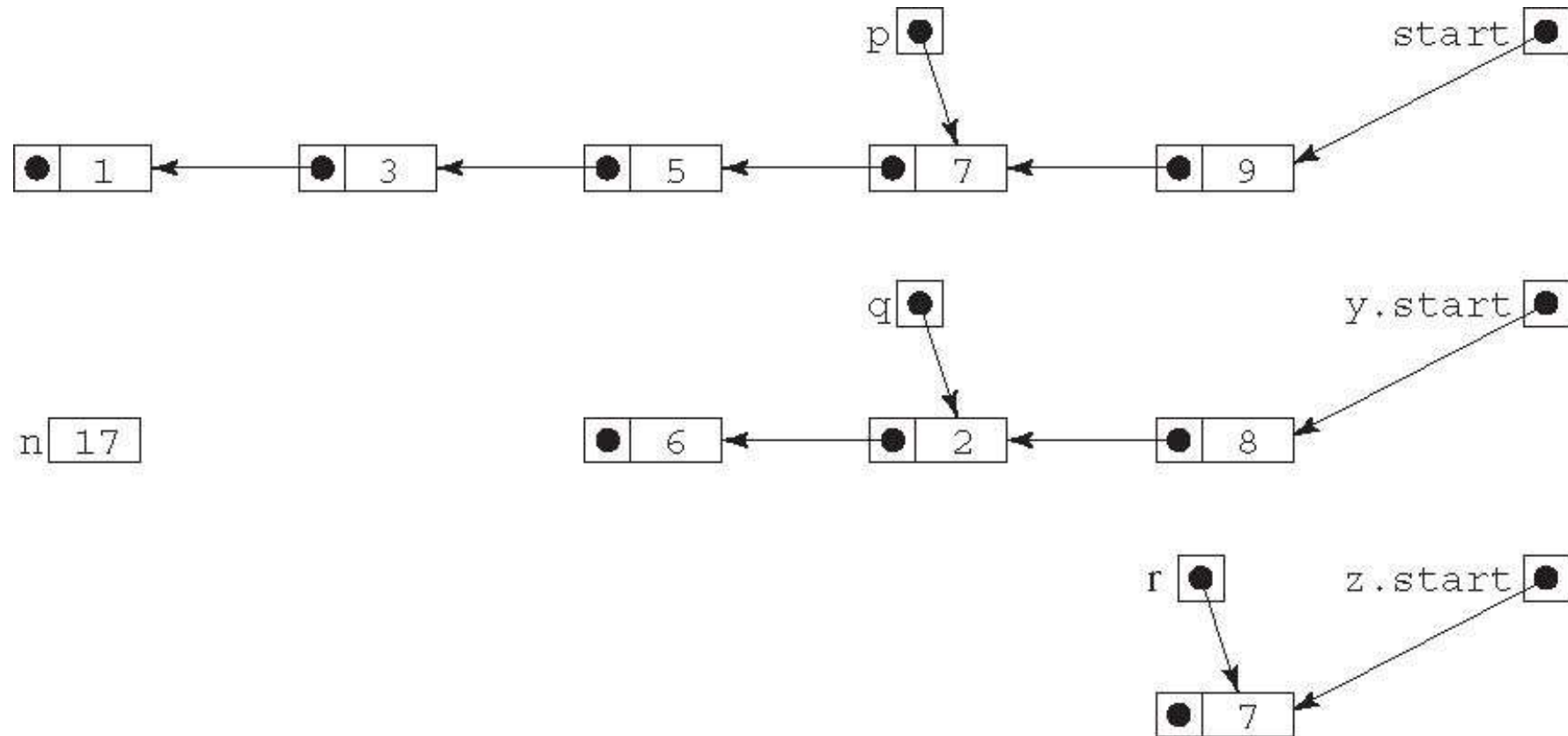
```
1 public BigInt(String s) {  
2     if (s.length() == 0)  
3         throw new IllegalArgumentException("empty string");  
4     start = new Node(digit(s, s.length()-1));  
5     Node p=start;  
6     for (int i = s.length()-2; i >= 0; i--)  
7         p = p.next = new Node(digit(s, i));  
8 }  
9  
10 private int digit(String s, int i) {  
11     String ss = s.substring(i, i+1);  
12     return Integer.parseInt(ss);  
13 }
```

피보나치 수의 생성에 의한 BigInt 클래스의 테스트

- LISTING 4.21: Testing the BigInt Class by Generating Fibonacci Numbers

```
1 public class TestBigInt {  
2     public static void main(String[] args) {  
3         BigInt x = new BigInt(0);  
4         BigInt y = new BigInt(1);  
5         BigInt z = new BigInt(1);  
6         for (int i = 0; i < 100; i++) {  
7             x = y;  
8             y = z;  
9             z = x.plus(y);  
10            System.out.println(z);  
11        }  
12    }  
13 }
```


BigInt 628과 BigInt 13,579의 덧셈



리스트 4.20: 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;  
6     p = p.next;  
7     q = q.next;  
8     while (p != null && q != null) {  
9         n = n/10 + p.digit + q.digit;  
10        r.next = new Node(n%10);  
11        p = p.next;  
12        q = q.next;  
13        r = r.next;  
14    }
```

```
15     while (p != null) {
16         n = n/10 + p.digit;
17         r.next = new Node(n%10);
18         p = p.next;
19         r = r.next;
20     }
21     while (q != null) {
22         n = n/10 + q.digit;
23         r.next = new Node(n%10);
24         q = q.next;
25         r = r.next;
26     }
27     if (n > 9)
28         r.next = new Node(n/10);
28     return z;
29 }
```