

# Chapter 13. 버퍼

**Originally made by Prof. Hanku Lee**

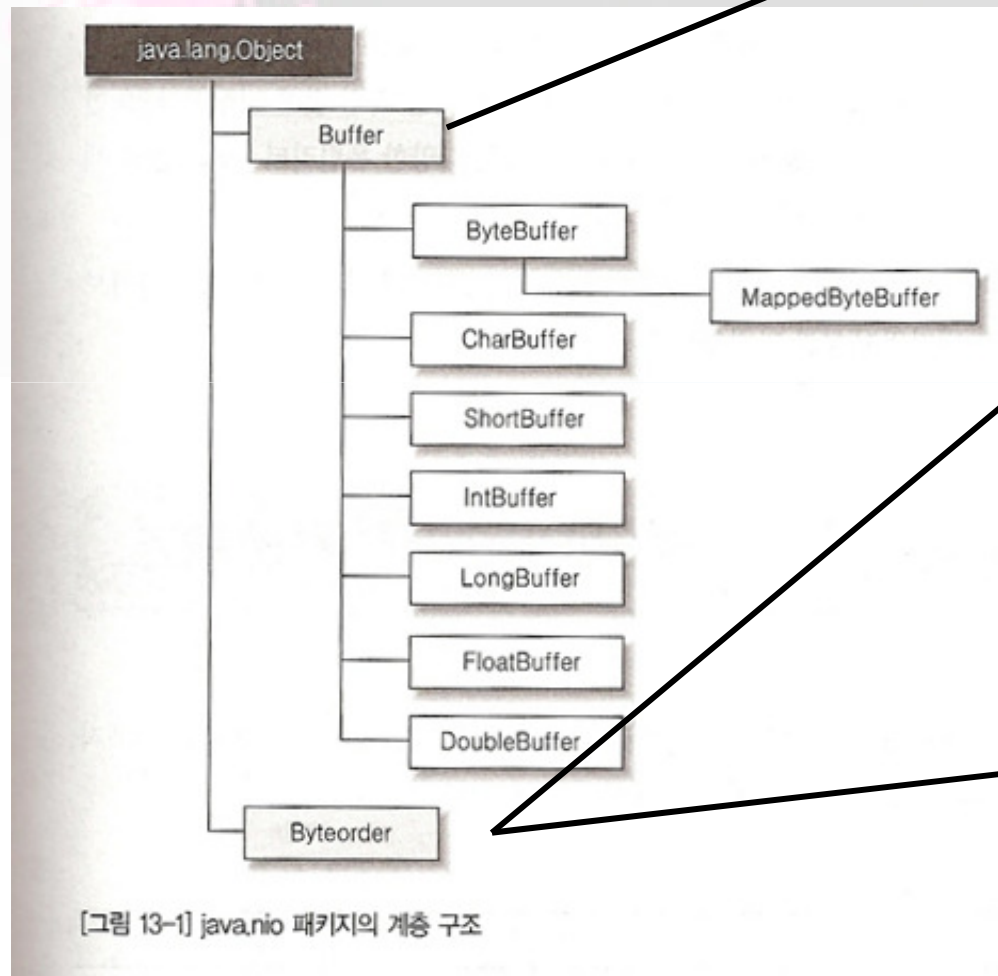
**Modified by Mingyu Lim**

**Collaborative Computing Systems Lab.  
School of Internet & Multimedia Engineering  
Konkuk University, Seoul, Korea**

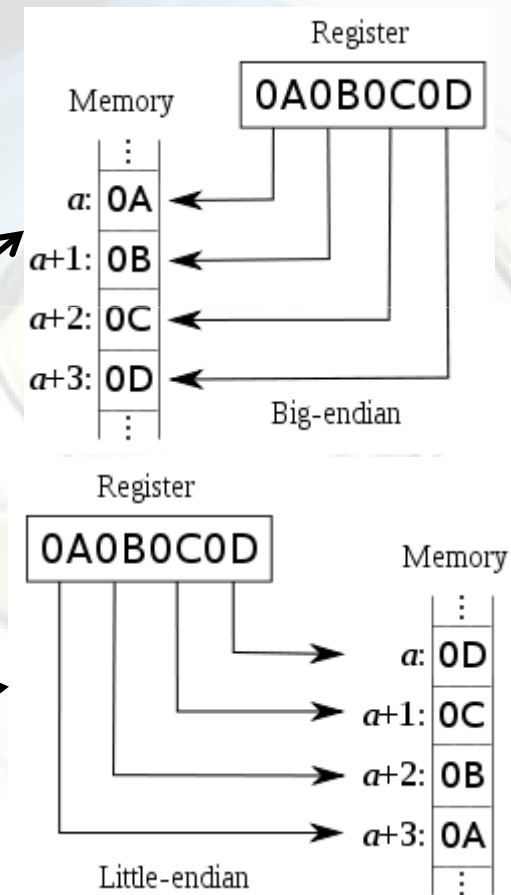
# 1. java.nio 패키지의 계층구조

## □ java.nio 패키지 구조

어떤 하나의 데이터형태들을 저장하는 컨테이너



[그림 13-1] java.nio 패키지의 계층 구조



## 2. 버퍼의 4가지 기본속성

### □ 4가지 기본 속성

속 성	설 명
position	버퍼에서 현재 읽거나 쓸 위치 값이다. limit보다 큰값을 가질수 없다. 즉 limit와 같은 값이 되면 더 이상 읽거나 쓰지 못한다는 의미고 만약, 읽거나 쓰기 작업을 할 경우 런타임 예외가 발생한다. 버퍼를 생성했을 때의 초기값은 0이다.
limit	버퍼에서 읽거나 쓸 수 있는 한계 값이다. 버퍼에서 실제 어디까지를 사용할지를 지정하는 속성 값으로 capacity 보다 클수 없다. 버퍼를 생성했을 때의 초기 값은 capacity와 같다.
capacity	버퍼의 크기를 나타낸다. 즉 메모리 크기라고 생각하면 된다. 이 값은 버퍼를 생성할때 파라미터로 주어진다. 한번 생성되면 크기를 변경할 수 없으므로 버퍼의 크기를 신중하게 결정해야 한다.
mark	Mark() 메소드로 현재의 position을 표시 해둘때 사용한다. 나중에 reset() 메소드를 호출해서 mark 위치로 position을 바꿀수 있다.

### - 속성값의 관계

```
0 ← mark ← position ← limit ← capacity
```

## 2. 버퍼의 4가지 기본속성(계속)

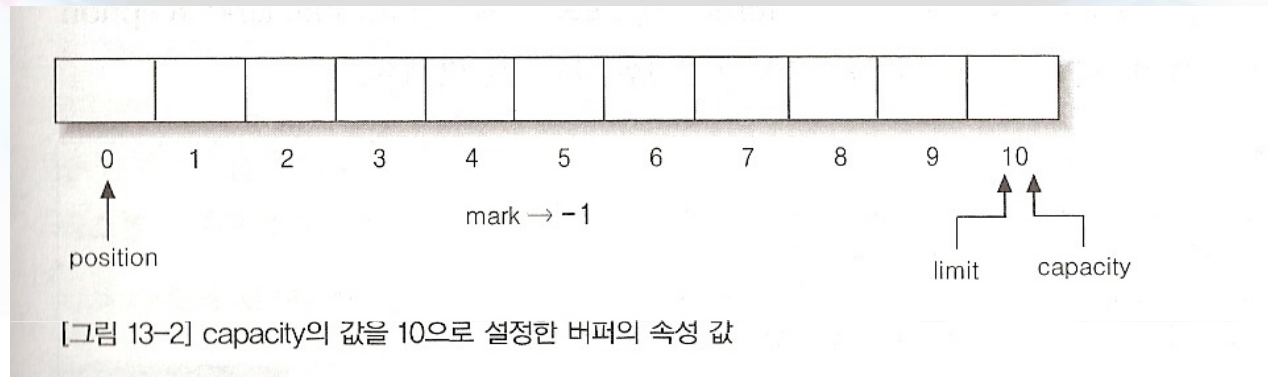
### □ Capacity 속성

- 버퍼의 바이트 크기를 정하는 것이 아니라 각 형식 버퍼에 맞는 기본 형식의 데이터를 각각의 기본형 버퍼 클래스에 몇 개나 넣을 수 있는지를 나타내는 것
- ByteBuffer의 capacity: 100
  - ◆ Byte형을 100개(100 byte) 넣을 수 있음
- IntBuffer의 capacity: 100
  - ◆ Int형을 100개 (400 byte) 넣을 수 있음
- LongBuffer의 capacity: 100
  - ◆ Long형을 100개 (800 byte) 넣을 수 있음

ACHROMATIC COLOR

## 2. 버퍼의 4가지 기본속성(계속)

### □ 4가지 기본 속성



- capacity가 10인 버퍼 생성 (그림 13-2) ,position 은 0으로 설정
- limit와 capacity 는 10으로 설정된다.

## 2. 버퍼의 4가지 기본속성(계속)

### □ Buffer 클래스의 기본 속성 제어 메소드

```
Public final int position();  
Public final int limit();  
Public final int capacity();  
  
Public final Buffer position(int newPosition);  
Public final Buffer limit(int newLimit);  
  
Public final Buffer mark();  
Public final Buffer reset();  
  
Public final int remaining(); // limit-position  
Public final boolean hasRemaining();  
Public abstract boolean isReadOnly();
```

ACHROMATIC COLOR  
\*\*\*\*\*

### 3. 버퍼에서 데이터 읽고 쓰기

#### □ ByteBuffer 메소드

```
// 상대적 위치로 읽고 쓰기
public abstract byte get();
public abstract ByteBuffer put (byte b);

// 절대적 위치로 읽고 쓰기
public abstract byte get (int index);
public abstract ByteBuffer put(int index, byte b);

// 버퍼의 데이터를 주어진 배열로 읽고 쓰기
public ByteBuffer get(byte[] dst);
public ByteBuffer get(byte[] dst, int offset, int length);
public final ByteBuffer put(byte[] src);
public ByteBuffer put (byte[] src, int offset, int length);

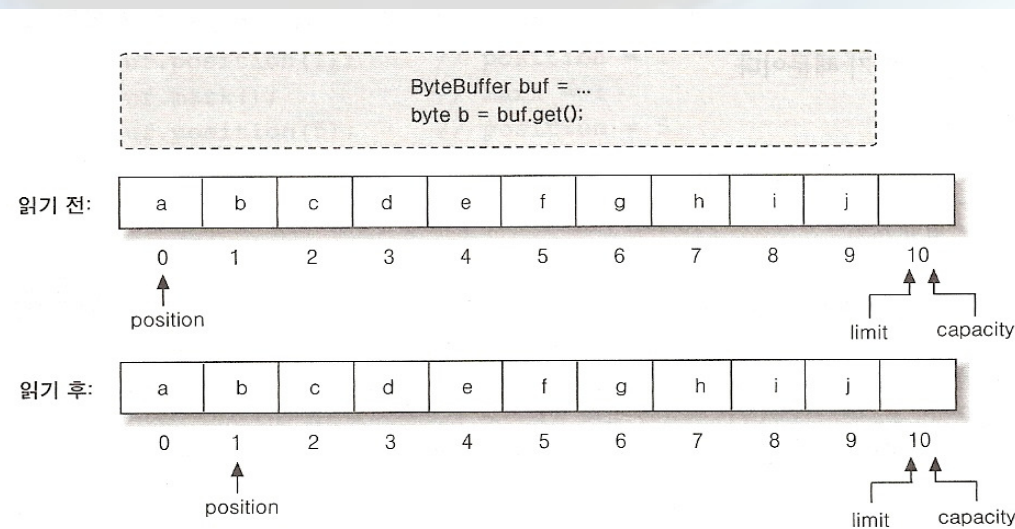
// 파라미터로 주어진 버퍼의 내용을 쓰기
public ByteBuffer put (ByteBuffer src);
```

1. 상대적 위치를 이용해서 1바이트씩 읽고 쓰기
2. 절대적 위치를 이용해서 1바이트씩 읽고 쓰기
3. 배열을 이용해서 한꺼번에 많은 데이터를 읽고 쓰기
4. 버퍼 자체를 파라미터로 받아서 쓰기

### 3. 버퍼에서 데이터 읽고 쓰기(계속)

#### □ 상대적 위치를 이용해서 1바이트씩 읽고 쓰기

- get() 메소드로 버퍼에서 데이터 읽기



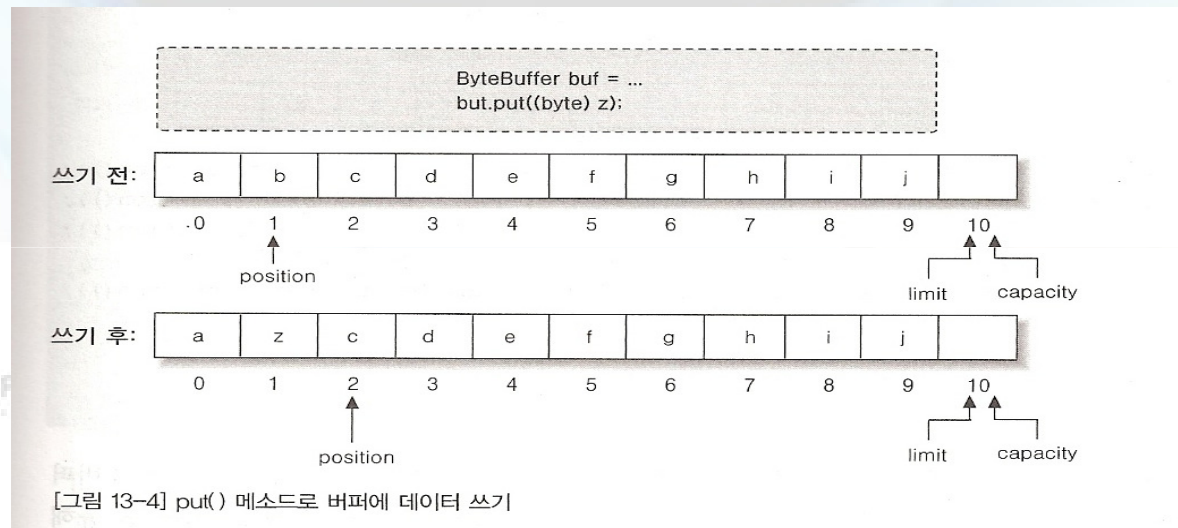
[그림 13-3] get() 메소드로 버퍼에서 데이터 읽기



### 3. 버퍼에서 데이터 읽고 쓰기(계속)

#### □ 상대적 위치를 이용해서 1바이트씩 읽고 쓰기

- put() 메소드로 버퍼에서 데이터 쓰기



- position 10인 상태에서 get(), put() 메소드 호출시  
BufferUnderflowException, BufferOverflowException 발생
- 읽고 쓰기 전에 항상 현재 position과 limit 비교한후 사용

### 3. 버퍼에서 데이터 읽고 쓰기(계속)

#### □ 상대적 위치를 이용해서 1바이트씩 읽고 쓰기 (예제)

```
RelativeBufferTest.java
import java.nio.ByteBuffer;

public class RelativeBufferTest {

    public static void main(String[] args) throws Exception {
        // 크기가 10인 ByteBuffer 를 생성.
        ByteBuffer buf = ByteBuffer.allocate(10);
        System.out.print("Init Position : " + buf.position());
        System.out.print(", Init Limit : " + buf.limit());
        System.out.println(", Init Capacity : " + buf.capacity());

        // 현재 position 이 0 인데 이곳에 mark 해둠.
        buf.mark();
        // a, b c 를 순서대로 버퍼에 넣는다.
        buf.put((byte) 10).put((byte) 11).put((byte) 12);
        // mark 해둔 0 인덱스로 position 을 되돌림.
        buf.reset();

        // 현재 position 의 버퍼에 있는 데이터를 출력함.
        System.out.println("Value : " + buf.get() + ", Position : " + buf.position());
        System.out.println("Value : " + buf.get() + ", Position : " + buf.position());
        System.out.println("Value : " + buf.get() + ", Position : " + buf.position());
        // position 4 에는 아무값도 넣지 않았지만 기본적으로 0이 입력됨을 볼 수 있다.
        System.out.println("Value : " + buf.get() + ", Position : " + buf.position());
    }
}
```

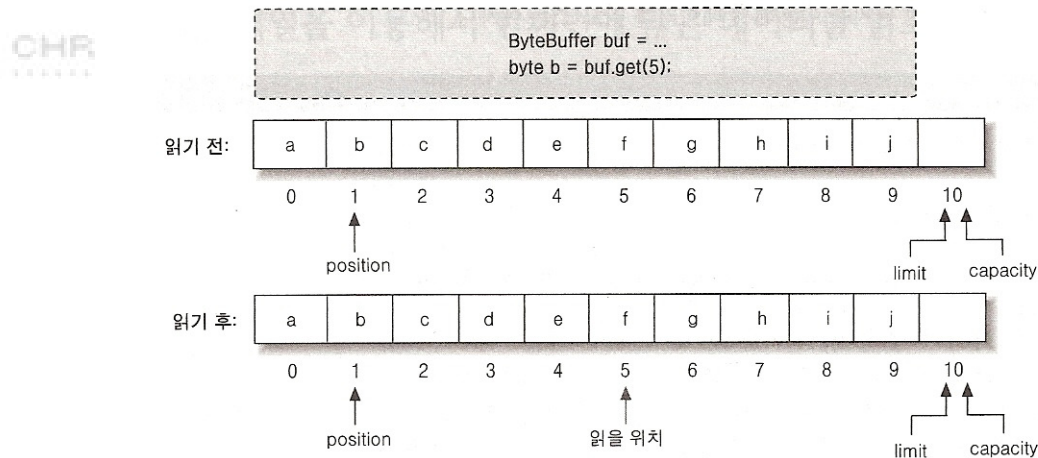
```
Console
<terminated> RelativeBufferTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 10. 09 오후 4:21:57)
Init Position : 0, Init Limit : 10, Init Capacity : 10
Value : 10, Position : 1
Value : 11, Position : 2
Value : 12, Position : 3
Value : 0, Position : 4
```

### 3. 버퍼에서 데이터 읽고 쓰기(계속)

#### □ 절대적 위치를 이용해서 1바이트씩 읽고 쓰기

```
// 절대적 위치로 읽고 쓰기
public abstract byte get (int index);
public abstract ByteBuffer put(int index, byte b);
```

- 상대적 방법은 현재의 position 위치 이용하는 반면 절대적 방법은 읽고 쓸 위치를 직접 지정
- 읽고 쓸때 position의 변화가 없다.
- get(int index) 메소드로 버퍼에서 데이터 읽기

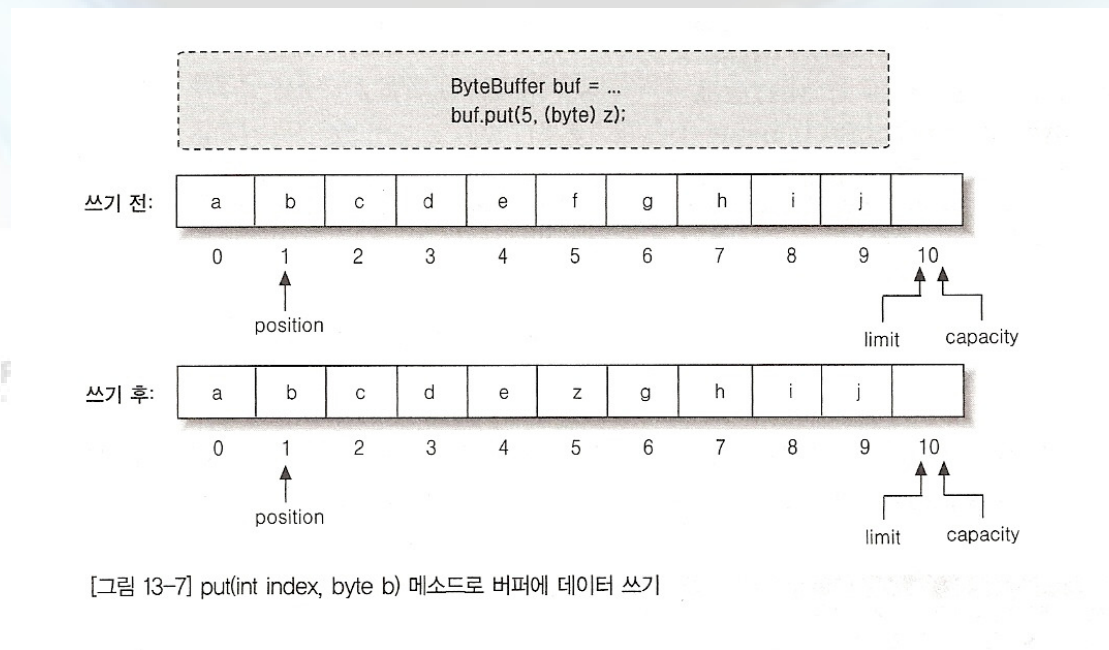


[그림 13-6] get(int index) 메소드로 버퍼에서 데이터 읽기

### 3. 버퍼에서 데이터 읽고 쓰기(계속)

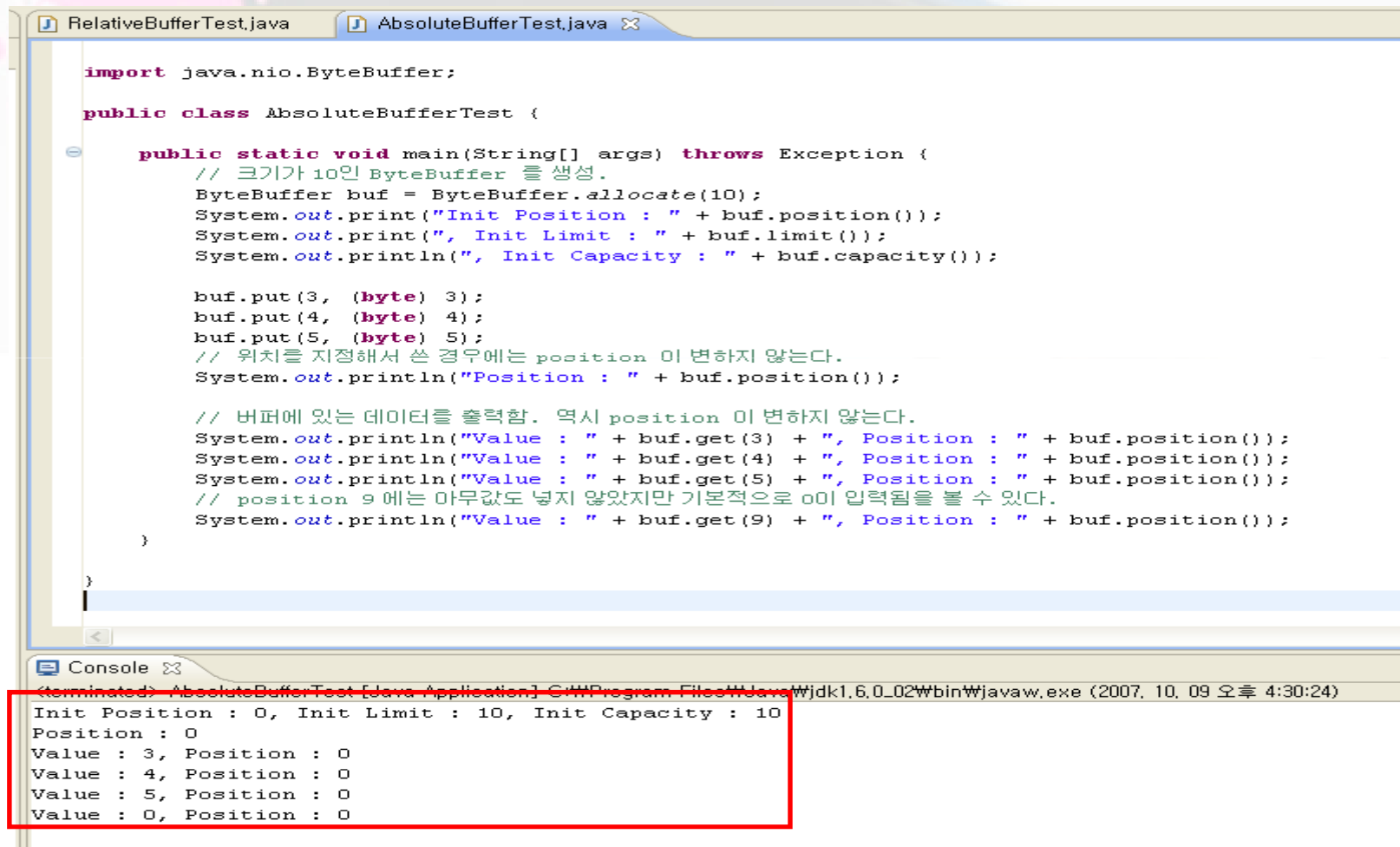
#### □ 절대적 위치를 이용해서 1바이트씩 읽고 쓰기

- put(int index,, byte b)메소드로 버퍼에 데이터 쓰기



### 3. 버퍼에서 데이터 읽고 쓰기(계속)

#### □ 절대적 위치를 이용해서 1바이트씩 읽고 쓰기(예제)



```
import java.nio.ByteBuffer;

public class AbsoluteBufferTest {

    public static void main(String[] args) throws Exception {
        // 크기가 10인 ByteBuffer 를 생성.
        ByteBuffer buf = ByteBuffer.allocate(10);
        System.out.print("Init Position : " + buf.position());
        System.out.print(", Init Limit : " + buf.limit());
        System.out.println(", Init Capacity : " + buf.capacity());

        buf.put(3, (byte) 3);
        buf.put(4, (byte) 4);
        buf.put(5, (byte) 5);
        // 위치를 지정해서 쓴 경우에는 position 이 변하지 않는다.
        System.out.println("Position : " + buf.position());

        // 버퍼에 있는 데이터를 출력함. 역시 position 이 변하지 않는다.
        System.out.println("Value : " + buf.get(3) + ", Position : " + buf.position());
        System.out.println("Value : " + buf.get(4) + ", Position : " + buf.position());
        System.out.println("Value : " + buf.get(5) + ", Position : " + buf.position());
        // position 9 에는 아무값도 넣지 않았지만 기본적으로 0이 입력됨을 볼 수 있다.
        System.out.println("Value : " + buf.get(9) + ", Position : " + buf.position());
    }
}
```

Console

(terminated) AbsoluteBufferTest [Java Application] C:\Program Files\Java\jdk1.6.0\_02\bin\javaw.exe (2007. 10. 09 오후 4:30:24)

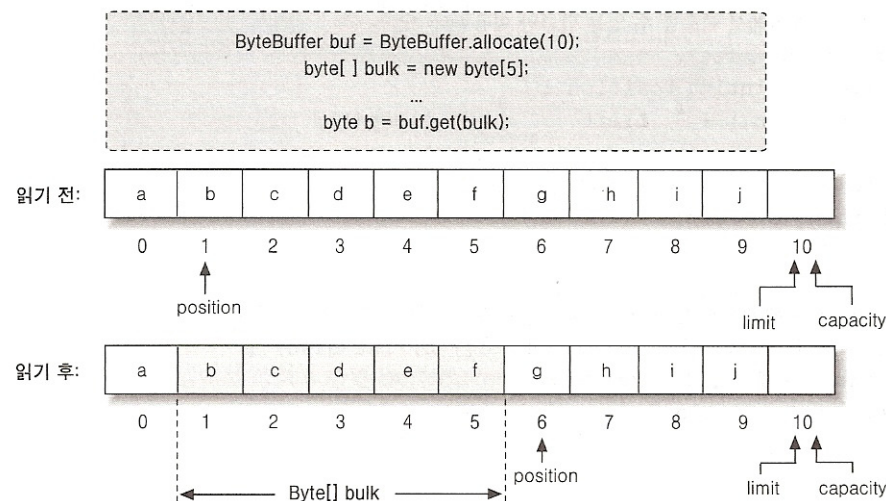
```
Init Position : 0, Init Limit : 10, Init Capacity : 10
Position : 0
Value : 3, Position : 0
Value : 4, Position : 0
Value : 5, Position : 0
Value : 0, Position : 0
```

### 3. 버퍼에서 데이터 읽고 쓰기(계속)

#### □ 배열을 이용해서 한꺼번에 많은 데이터를 읽고 쓰기

```
// 버퍼의 데이터를 주어진 배열로 읽고 쓰기
public ByteBuffer get(byte[] dst);
public ByteBuffer get(byte[] dst, int offset, int length);
public final ByteBuffer put(byte[] src);
public ByteBuffer put (byte[] src, int offset, int length);
```

#### • 버퍼의 데이터를 배열로 읽어 들이기

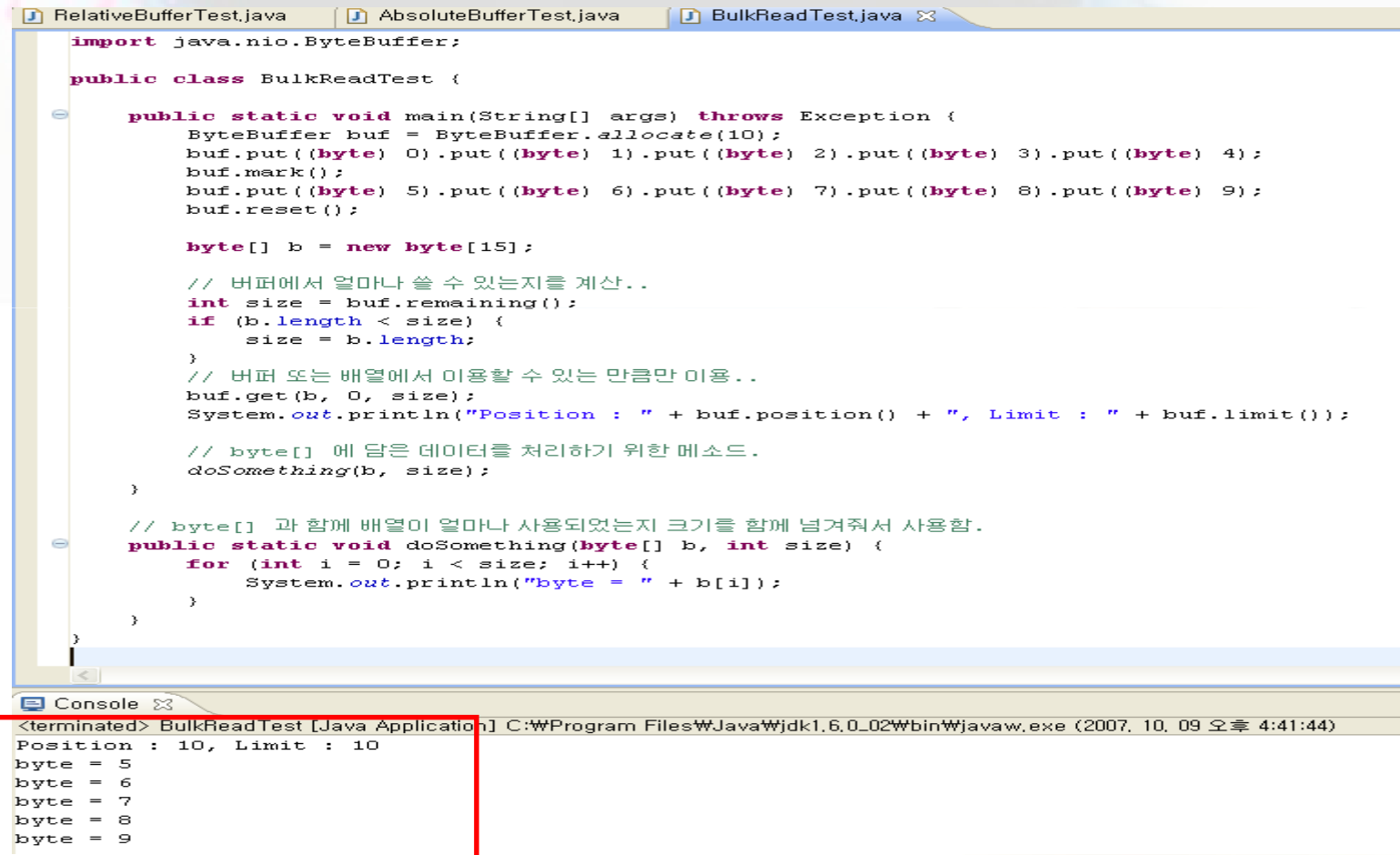


[그림 13-9] 버퍼의 데이터를 배열로 읽어 들이기

1. `get(bulk)` 는 `get(bulk, 0, bulk.length)` 같은 의미
2. Byte 버퍼 bulk에 버퍼의 인덱스 1부터 5까지 데이터가 입력
3. 상대적 방법과 같이 position 위치변경
4. 버퍼의 limit을 넘어서 데이터를 읽으려하면 `BufferUnderflowException` 발생

### 3. 버퍼에서 데이터 읽고 쓰기(계속)

- 배열을 이용해서 한꺼번에 많은 데이터를 읽고 쓰기
  - 버퍼의 데이터를 배열로 읽어 들이기(예제)



```
import java.nio.ByteBuffer;

public class BulkReadTest {

    public static void main(String[] args) throws Exception {
        ByteBuffer buf = ByteBuffer.allocate(10);
        buf.put((byte) 0).put((byte) 1).put((byte) 2).put((byte) 3).put((byte) 4);
        buf.mark();
        buf.put((byte) 5).put((byte) 6).put((byte) 7).put((byte) 8).put((byte) 9);
        buf.reset();

        byte[] b = new byte[15];

        // 버퍼에서 얼마나 쓸 수 있는지를 계산..
        int size = buf.remaining();
        if (b.length < size) {
            size = b.length;
        }
        // 버퍼 또는 배열에서 이용할 수 있는 만큼만 이용..
        buf.get(b, 0, size);
        System.out.println("Position : " + buf.position() + ", Limit : " + buf.limit());

        // byte[] 에 담은 데이터를 처리하기 위한 메소드.
        doSomething(b, size);
    }

    // byte[] 과 함께 배열이 얼마나 사용되었는지 크기를 함께 넘겨줘서 사용함.
    public static void doSomething(byte[] b, int size) {
        for (int i = 0; i < size; i++) {
            System.out.println("byte = " + b[i]);
        }
    }
}
```

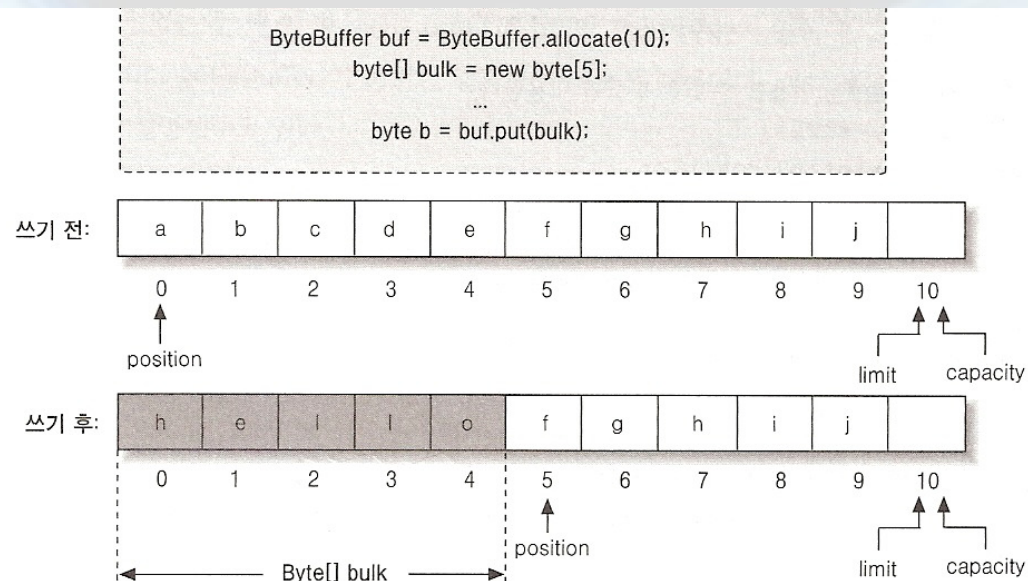
Console

```
<terminated> BulkReadTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 10. 09 오후 4:41:44)
Position : 10, Limit : 10
byte = 5
byte = 6
byte = 7
byte = 8
byte = 9
```



### 3. 버퍼에서 데이터 읽고 쓰기(계속)

- 배열을 이용해서 한꺼번에 많은 데이터를 읽고 쓰기
  - 배열안의 데이터를 버퍼에 쓰기



[그림 13-11] 배열 안의 데이터를 버퍼에 쓰기



### 3. 버퍼에서 데이터 읽고 쓰기(계속)

- 배열을 이용해서 한꺼번에 많은 데이터를 읽고 쓰기
  - 배열안의 데이터를 버퍼에 쓰기(예제)

```
import java.nio.ByteBuffer;

public class BulkWriteTest {

    public static void main(String[] args) throws Exception {
        ByteBuffer buf = ByteBuffer.allocate(10);
        buf.position(5);
        buf.mark();
        System.out.println("Position : " + buf.position() + ", Limit : " + buf.limit());

        byte[] b = new byte[15];
        for (int i = 0; i < b.length; i++) {
            b[i] = (byte) i;
        }

        // 버퍼에서 얼마나 쓸 수 있는지를 계산..
        int size = buf.remaining();
        if (b.length < size) {
            size = b.length;
        }
        // 버퍼 또는 배열에서 이용할 수 있는 만큼만 이용..
        buf.put(b, 0, size);
        System.out.println("Position : " + buf.position() + ", Limit : " + buf.limit());

        // byte[] 에 담은 데이터를 처리하기 위한 메소드.
        buf.reset();
        doSomething(buf, size);
    }

    // byte[] 과 함께 배열이 얼마나 사용되었는지 크기를 함께 넘겨줘서 사용함.
    public static void doSomething(ByteBuffer buf, int size) {
        for (int i = 0; i < size; i++) {
            System.out.println("byte = " + buf.get());
        }
    }
}
```

Console

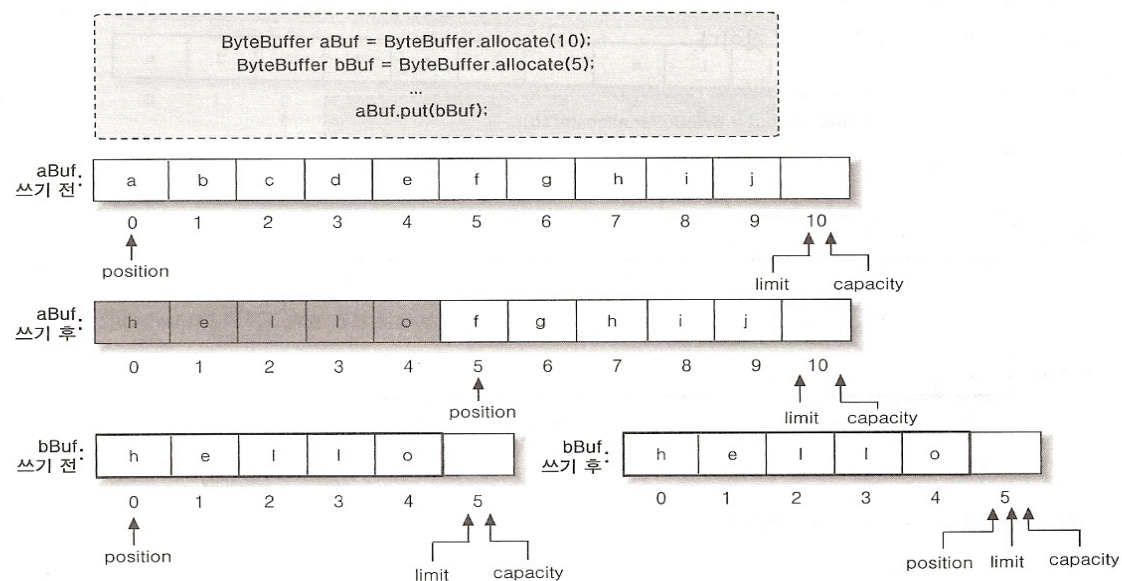
```
<terminated> BulkWriteTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 10. 09 오후 4:44:58)
Position : 5, Limit : 10
Position : 10, Limit : 10
byte = 0
byte = 1
byte = 2
byte = 3
byte = 4
```

### 3. 버퍼에서 데이터 읽고 쓰기(계속)

#### □ 버퍼자체를 파라미터로 받아서 쓰기

```
// 파라미터로 주어진 버퍼의 내용을 쓰기  
public ByteBuffer put (ByteBuffer src);
```

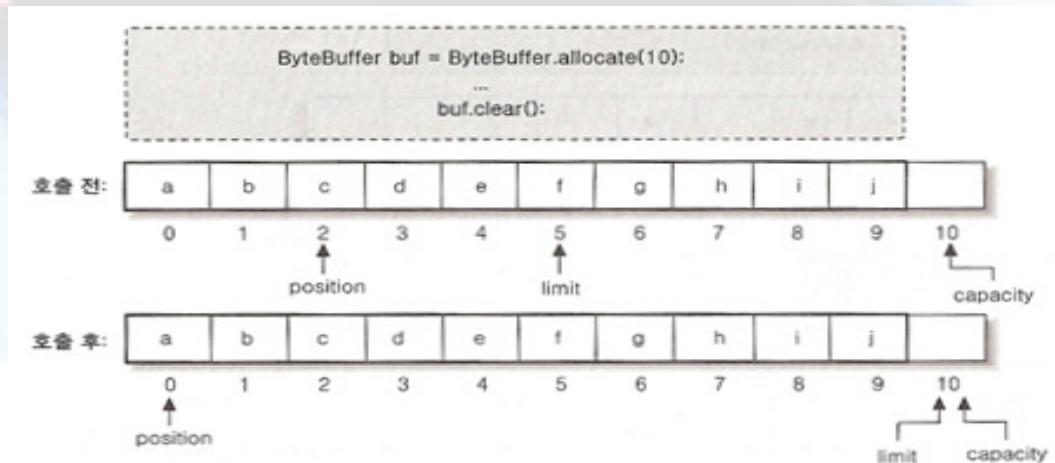
#### • 파라미터로 주어진 버퍼의 내용을 쓰기



[그림 13-13] 파라미터로 주어진 버퍼의 내용을 쓰기

## 4. Buffer클래스가 제공하는 유틸리티 메소드

### □ clear( ) 메소드

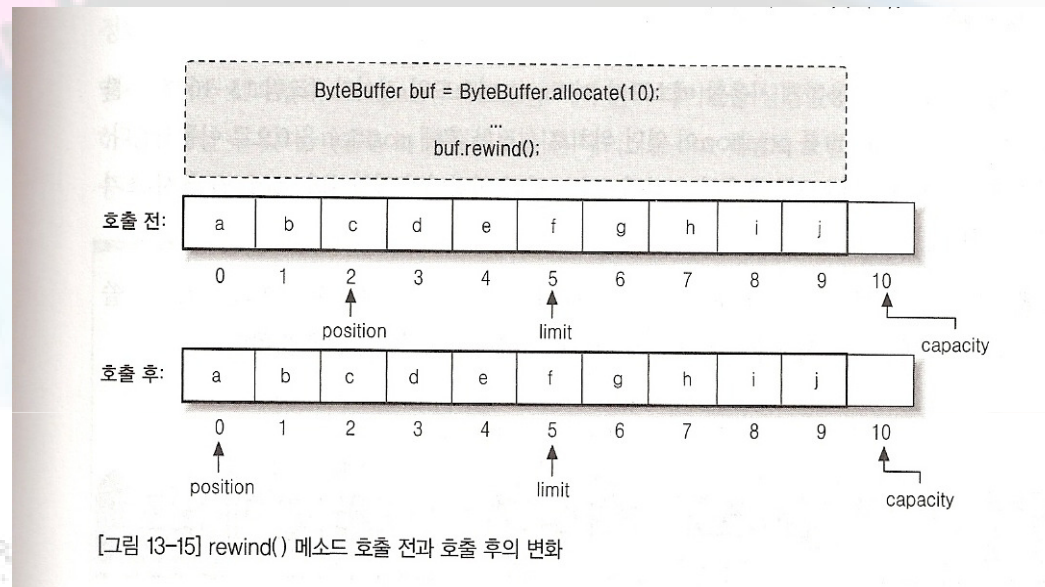


CHROF [그림 13-14] clear() 메소드 호출 전과 호출 후의 변화

- 메소드 호출전에는 position 과 limit이 각각 2와 5였지만 clear( ) 메소드 호출후 0 과 10으로 변경
- 호출되기전에 mark 값이 설정되었다면 mark값 역시 -1로 초기화
- 벡터나 해시테이블같은 컬렉션 클래스들의 clear()와 달리 버퍼 클래스는 각 속성의 값만 초기화시킨다.

## 4. Buffer클래스가 제공하는 유틸리티 메소드

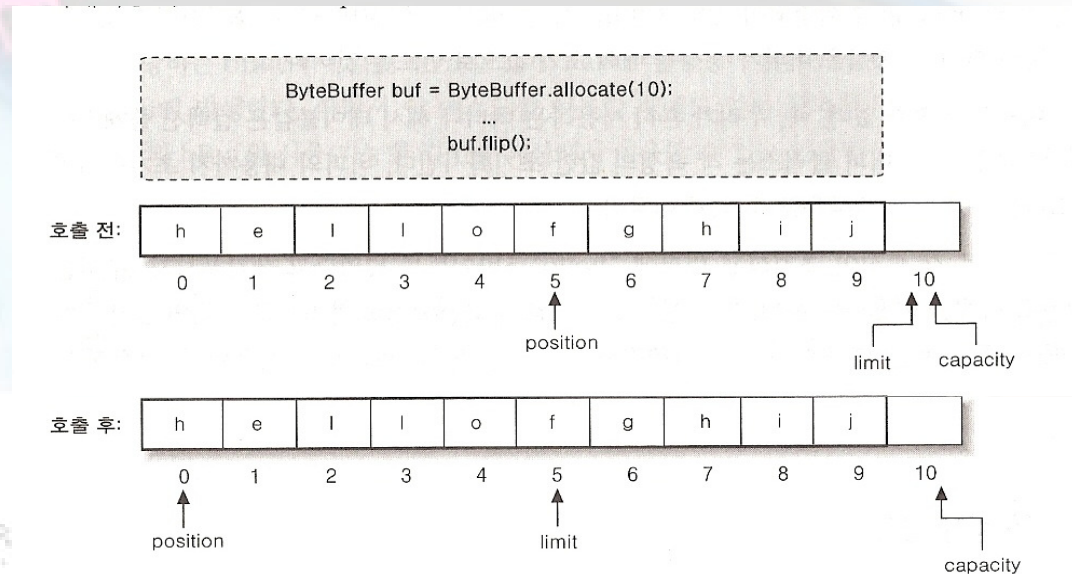
### □ rewind( ) 메소드



- position 속성만 초기화
- 호출되기전에 mark 값이 설정되었다면 mark값 역시 -1로 초기화
- 읽었던 데이터를 다시 읽기 위해 사용

## 4. Buffer클래스가 제공하는 유틸리티 메소드

### □ flip( ) 메소드

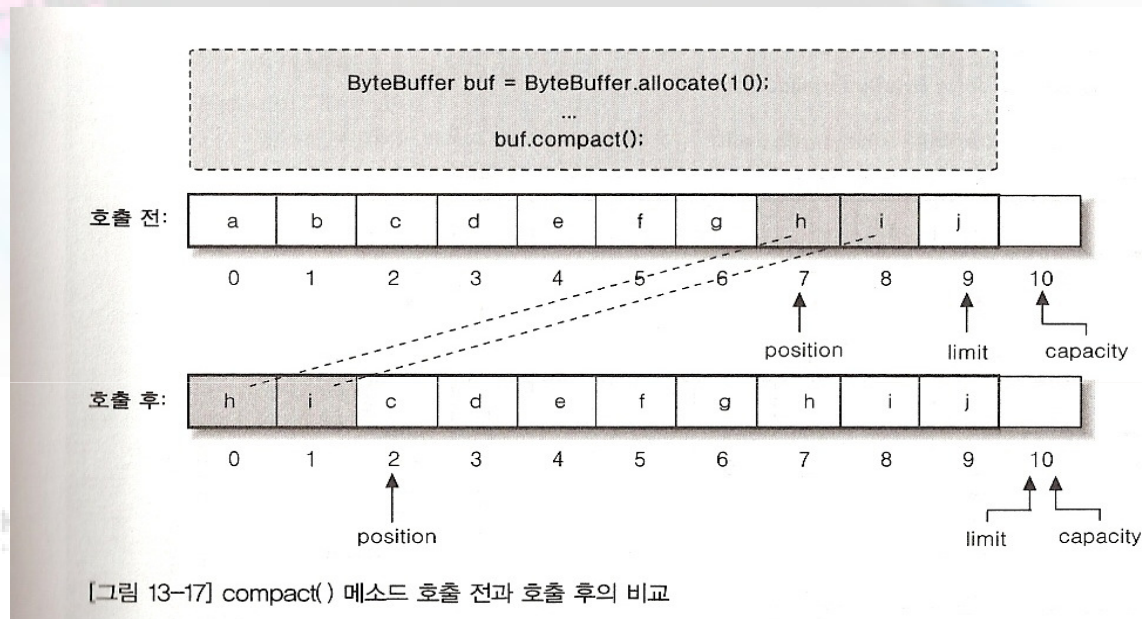


[그림 13-16] flip( ) 메소드 호출 전과 호출 후의 비교

- limit를 position이 있던 위치로 설정한후에 position을 0으로 이동
- 호출되기전에 mark 값이 설정되었다면 mark값 역시 -1로 초기화
- 이미 사용했던 버퍼를 재사용하기 위해 버퍼의 clear() 메소드를 호출하고 데이터를 버퍼에 쓴 후, flip()메소드로 재사용할 범위를 지정한 후 데이터 읽음

## 5. Buffer 하위 클래스 유틸리티 메소드

### □ compact( ) 메소드

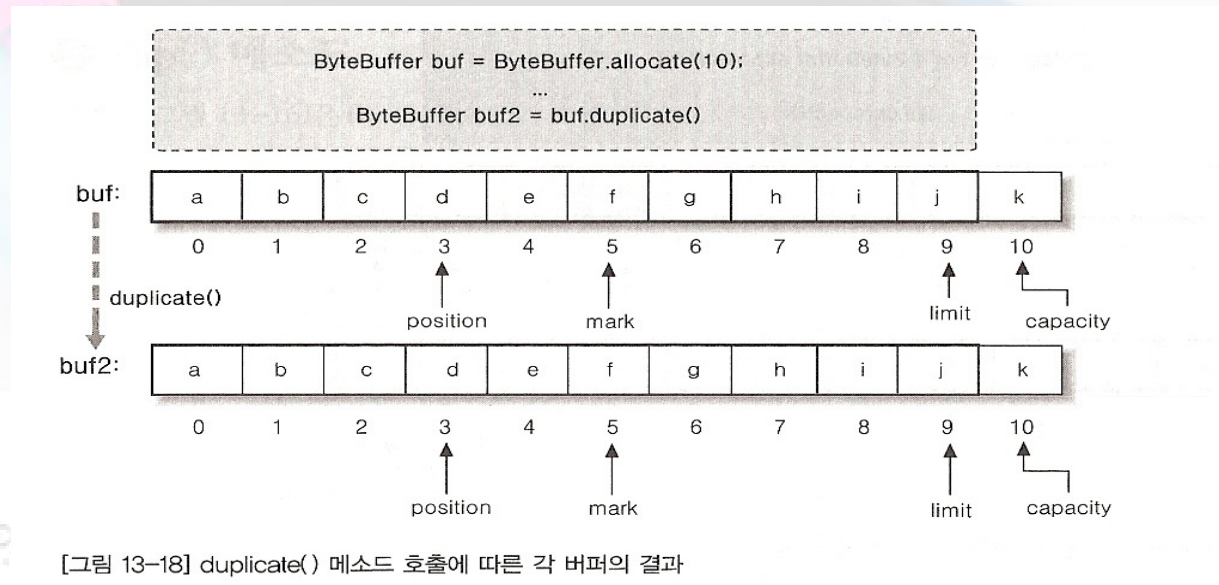


- position에서 limit사이의 남아있는 데이터를 버퍼의 맨앞에서부터 쓴다.
- 쓴만큼 position 이동시키고, limit는 초기화
- 호출되기전에 mark 값이 설정되었다면 mark값 역시 -1로 초기화
- 버퍼 안의 데이터를 남김없이 모두 전송하고 싶을 때 유용하게 사용



## 5. Buffer 하위 클래스 유틸리티 메소드(계속)

### □ duplicate( ) 메소드



- 버퍼에 복사본을 생성할 때 사용하는 메소드
- 데이터와 기본 속성들을 포함한 새로운 버퍼를 생성하는것이 아니고 단지 원본 버퍼와 같은 메모리 공간을 참조하는 버퍼 생성
- 버퍼에 저장된 데이터는 같은 공간을 참조하고 있기 때문에 어느 한쪽에 에서 데이터를 수정하면 원본과 복사본 버퍼 모두에 반영

## 예제 13-5 DuplicateTest

```
public class DuplicateTest {
    public static void main(String[] args) {
        // 크기가 10인 ByteBuffer 생성
        ByteBuffer buf = ByteBuffer.allocate(10);
        // 버퍼의 position 인덱스와 같은 값을 넣음
        buf.put((byte) 0).put((byte) 1).put((byte) 2).put((byte) 3).put((byte) 4)
            .put((byte) 5).put((byte) 6).put((byte) 7).put((byte) 8).put((byte) 9);
        // position 을 3으로 변경
        buf.position(3);
        // limit 를 9로 변경
        buf.limit(9);
        // 현재 position 3 을 마크해둠
        buf.mark();

        // 원래 버퍼의 복사본 생성
        ByteBuffer buf2 = buf.duplicate();
        // 복사된 버퍼의 position, limit, capacity를 출력
        System.out.println(
            "1) Position: " + buf2.position()
            + ", Limit: " + buf2.limit()
            + ", Capacity: " + buf2.capacity()
        );
        // position 을 7로 변경
        buf2.position(7);
        buf2.reset();
        System.out.println("reset() 호출 후 Position: " + buf2.position());
    }
}
```

1) Position: 3, Limit: 9, Capacity: 10  
reset() 호출 후 Position: 3



## 예제 13-5 DuplicateTest (계속)

```
// buf2 를 clear 함
buf2.clear();
// clear 한 후의 복사된 버퍼의 position, limit, capacity를 출력
System.out.println(
    "2) Position: " + buf2.position()
    + ", Limit: " + buf2.limit()
    + ", Capacity: " + buf2.capacity());

// 복사된 버퍼의 내용을 출력
while (buf2.hasRemaining()) {
    System.out.print(buf2.get() + " ");
}

// 원래 버퍼에 0번째 값을 바꿈
buf.put(0, (byte) 10);
System.out.println("\n" + "=> buf 의 0 값을 10 으로 바꿈");

// 원래 버퍼와 복사된 버퍼에서 값이 변경되었는지 확인 위해 출
System.out.println("Original Buffer Value : " + buf.get(0));
System.out.println("DuplicateBuffer Value : " + buf2.get(0));

// 복사된 버퍼에 1번째 값을 바꿈
buf2.put(1, (byte) 11);
System.out.println("=> buf2 의 1 값을 12 로 바꿈");

// 원래 버퍼와 복사된 버퍼에서 값이 변경되었는지 확인 위해 출
System.out.println("Original Buffer Value : " + buf.get(1));
System.out.println("DuplicateBuffer Value : " + buf2.get(1));
}
}
```

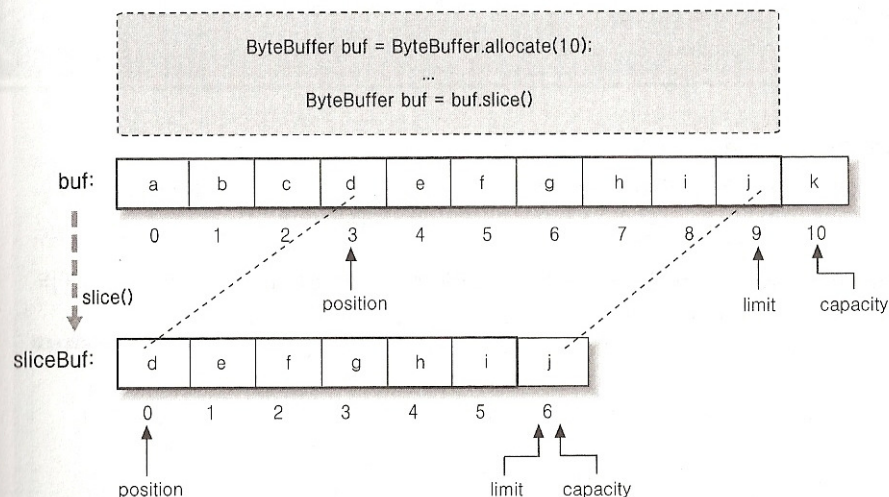
```
2) Position: 0, Limit: 10, Capacity: 10
0 1 2 3 4 5 6 7 8 9
=> buf 의 0 값을 10 으로 바꿈
Original Buffer Value : 10
DuplicateBuffer Value : 10
=> buf2 의 1 값을 12 로 바꿈
Original Buffer Value : 11
DuplicateBuffer Value : 11
```

## 5. Buffer 하위 클래스 유틸리티 메소드(계속)

### ❑ asReadOnlyBuffer( ) 메소드

- duplicate() 메소드로 생성된 버퍼와 한가지 제외하고 동일
- duplicate() 메소드 호출로 생성된 버퍼는 put() 메소드의 호출로 데이터 수정할수 있었다. asReadOnlyBuffer 메소드로 생성된 버퍼는 단지 버퍼의 데이터를 읽을 수만 있다.

### ❑ slice( ) 메소드



[그림 13-20] slice() 메소드 호출에 따른 각 버퍼의 결과

- 버퍼의 일부분만 복사
- 원본 버퍼의 position에서 limit까지만 떼어내어 생성
- 역시 자신만의 네가지 속성을 갖고, 원본 버퍼와 동일한 메모리 구조 참조
- duplicate(), asReadOnlyBuffer()와 차이점: 기본 속성이 초기화 됨

## 예제 13-6 SliceTest

```
public class SliceTest {
    public static void main(String[] args) {
        // 크기가 10인 ByteBuffer 생성
        ByteBuffer buf = ByteBuffer.allocate(10);
        // 버퍼의 position 인덱스와 같은 값을 넣음
        buf.put((byte) 0).put((byte) 1).put((byte) 2).put((byte) 3).put((byte) 4)
            .put((byte) 5).put((byte) 6).put((byte) 7).put((byte) 8).put((byte) 9);
        // position 을 3으로 변경
        buf.position(3);
        // limit 를 9로 변경
        buf.limit(9);

        // 원래 버퍼의 일부분을 slice한 버퍼 생성
        ByteBuffer buf2 = buf.slice();
        // Slice한 버퍼의 position, limit, capacity를 출력
        System.out.println(
            "1) Position: " + buf2.position()
            + ", Limit: " + buf2.limit()
            + ", Capacity: " + buf2.capacity()
        );
    }
}
```

1) Position: 0, Limit: 6, Capacity: 6

## 예제 13-6 SliceTest (계속)

```
// slice한 버퍼의 내용을 출력
while (buf2.hasRemaining()) {
    System.out.print(buf2.get() + " ");
}

// 원래 버퍼에 3번째 값을 바꿈
buf.put(3, (byte) 10);
System.out.println("\n" + "=> buf 의 3 값을 10 으로 바꿈");

// 원래 버퍼와 slice한 버퍼에서 값이 변경되었는지 확인 위해 출
System.out.println("Original Buffer Value : " + buf.get(3));
System.out.println("SliceBuffer Value : " + buf2.get(0));

// 복사된 버퍼에 4번째 값을 바꿈
buf2.put(4, (byte) 11);
System.out.println("=> buf2 의 4 값을 11 로 바꿈");

// 원래 버퍼와 slice한 버퍼에서 값이 변경되었는지 확인 위해 출
System.out.println("Original Buffer Value : " + buf.get(7));
System.out.println("SliceBuffer Value : " + buf2.get(4));
}
}
```

```
3 4 5 6 7 8
=> buf 의 3 값을 10 으로 바꿈
Original Buffer Value : 10
SliceBuffer Value : 10
=> buf2 의 4 값을 11 로 바꿈
Original Buffer Value : 11
SliceBuffer Value : 11
```

## 6. 버퍼 만들기

### □ 버퍼 생성 방법

```
public abstract class ByteBuffer extends Buffer implements Comparable{
    //팩토리 메소드를 사용해서 만드는 방법
    public static ByteBuffer allocate(int capacity);

    //이미 존재하는 배열을 이용해서 만드는 방법
    public static ByteBuffer wrap(byte[] array);
    public static ByteBuffer wrap(byte[] array, int offset, int length);

    //유틸리티 메소드들
    public final boolean hasArray();
    public final byte[] array();
    public final int arrayOffset();
}
```

- allocate() 사용해서 버퍼를 생성하는 방법  
(바이트 전개가 저장될 버퍼를 생성)

```
ByteBuffer buf = ByteBuffer.allocate(1000);
```

## 6. 버퍼 만들기(계속)

### □ 버퍼 생성 방법

- wrap( ) 사용해서 버퍼를 생성하는 방법

```
byte[] myArray = new byte[1024];  
ByteBuffer buf = ByteBuffer.wrap(myArray);
```

(1024크기를 갖는 버퍼 생성, 이때 buf는 myArray 를 참조하기 때문에 myArray나 buf 중 하나를 변경하면 양쪽 모두에게 반영)

- wrap( ) 사용한 코드 템플릿

```
byte[] myArray = new byte [1024];  
  
ByteBuffer buf = ByteBuffer.wrap(myArray, 10, 100);
```

- position이 10, limit가 100으로 설정
- slice()와 달리 일부분을 잘라내어 생성하는 것이 아니라, myArray 전체를 생성하되 파라미터로 설정된 부분만 이용하도록 초기화됨
- clear()를 호출하면 myArray의 0부터 1024까지 접근 가능

## 6. 버퍼 만들기(계속)

### □ nondirect(nondirect) 버퍼

- Allocate(), wrap()을 이용해서 생성한 버퍼는 시스템 메모리가 아닌 JVM의 힙영역에 저장
- 내부적으로 해당 버퍼 형식의 보조배열 존재
  - ◆ hasArray() 호출하면 true 리턴
  - ◆ Array() 메소드로 보조배열 접근 가능
  - ◆ arrayOffset() 메소드로 보조배열내의 이 버퍼의 최초 시작위치 확인 가능

CHROMATIC COLOR  
.....

ACHROMATIC COLOR  
.....

## 7. ByteBuffer

### □ ByteBuffer

ByteBuffer는 시스템 메모리를 직접 사용하는 다이렉트 버퍼를 만들수 있는 유일한 버퍼 클래스기 때문에 가장 중요하다.

### □ 다이렉트 버퍼

- ByteBuffer만 다이렉트 버퍼를 만들수 있는 이유

CHROMATIC COLOR

1. 운영체제가 이용하는 가장 기본적인 데이터 단위가 바이트다.

2. 시스템 메모리 또한 순차적인 바이트들의 집합이기 때문

CHROMATIC COLOR



## 7. ByteBuffer(계속)

### □ 다이렉트 버퍼

#### - API 코드

```
public abstract class ByteBuffer extends Buffer implements Comparable{  
    public static ByteBuffer allocateDirect(int capacity);  
    public abstract boolean isDirect();  
}
```

#### - 코드 템플릿

```
ByteBuffer directBuffer = ByteBuffer.allocateDirect(1024);  
  
Boolean isDirect = directBuffer.isDirect();
```

## 7. ByteBuffer(계속)

### □ 다이렉트 버퍼

- 생성된 다이렉트 버퍼는 채널과 네이티브 IO 루틴들이 이용  
(효율적으로 IO 작업을 수행)
- 다이렉트 버퍼는 시스템 메모리를 가짐  
(JNI를 통해서 다이렉트 버퍼 생성, 조작, 가비지 컬렉션한다)
- `ByteBuffer.allocateDirect()` 이용 버퍼 생성하면 내부적으로 다음과 같이 작동  
: 메소드 파라미터로 주어진 크기만큼의 시스템 메모리를 JNI 사용해서 할당  
: 시스템 메모리를 래핑하는 자바 객체를 만들어 리턴
- 채널의 타겟이 되는 것은 다이렉트 버퍼만이 가능
- 다이렉트 버퍼를 메모리에 로드하거나 릴리즈 하는것은 언다이렉트 버퍼를 생성, 제거하는 것에 비해 상당히 부하가 크다.
- 성능에 민감하고 버퍼를 오랫동안 유지해서 사용할 필요가 있는곳에서 사용하는것이 바람직

## 7. ByteBuffer(계속)

### □ 뷰 버퍼

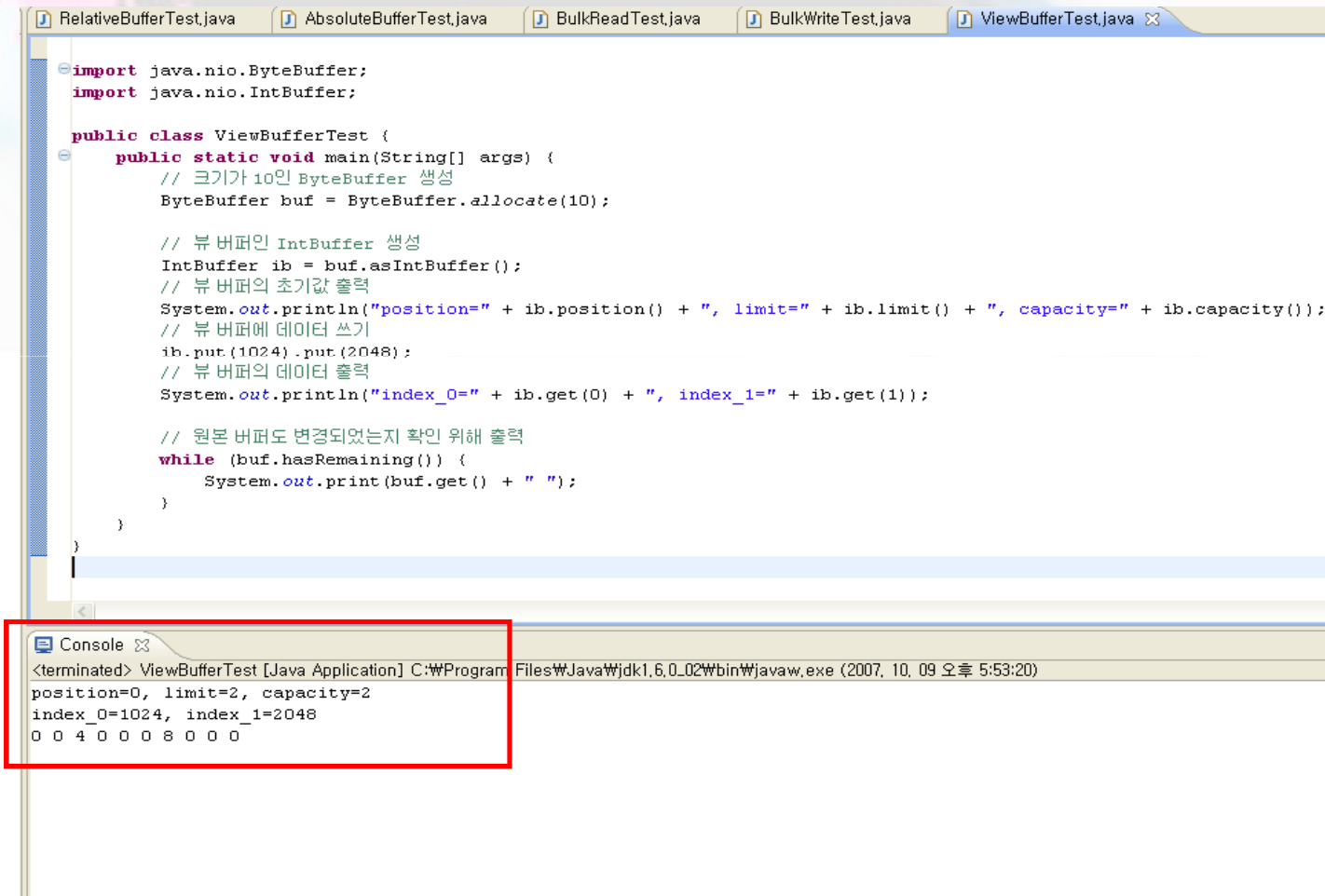
```
public abstract class ByteBuffer extends Buffer implements Comparable{  
    public abstract CharBuffer asCharBuffer();  
    public abstract ShortBuffer asShortBuffer();  
    public abstract asIntBuffer();  
    public abstract asLongBuffer();  
    public abstract asFloatBuffer();  
    public abstract asDoubleBuffer();  
}
```

- position, mark, limit, capacity 를 갖고 원본 버퍼와 데이터를 공유하는 즉 자신을 제외한 다른 여섯 가지 기본 형식의 뷰 버퍼를 생성할수 있는 팩토리 메소드를 제공

ACHROMATIC COLOR  
\*\*\*\*\*

## 7. ByteBuffer(계속)

### □ 뷰 버퍼(예제)



The screenshot shows a Java IDE with several tabs open: RelativeBufferTest.java, AbsoluteBufferTest.java, BulkReadTest.java, BulkWriteTest.java, and ViewBufferTest.java. The ViewBufferTest.java tab is active, displaying the following code:

```
import java.nio.ByteBuffer;
import java.nio.IntBuffer;

public class ViewBufferTest {
    public static void main(String[] args) {
        // 크기가 10인 ByteBuffer 생성
        ByteBuffer buf = ByteBuffer.allocate(10);

        // 뷰 버퍼인 IntBuffer 생성
        IntBuffer ib = buf.asIntBuffer();
        // 뷰 버퍼의 초기값 출력
        System.out.println("position=" + ib.position() + ", limit=" + ib.limit() + ", capacity=" + ib.capacity());
        // 뷰 버퍼에 데이터 쓰기
        ib.put(1024).put(2048);
        // 뷰 버퍼의 데이터 출력
        System.out.println("index_0=" + ib.get(0) + ", index_1=" + ib.get(1));

        // 원본 버퍼도 변경되었는지 확인 위해 출력
        while (buf.hasRemaining()) {
            System.out.print(buf.get() + " ");
        }
    }
}
```

Below the code editor, the console window is visible, showing the output of the program:

```
<terminated> ViewBufferTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 10. 09 오후 5:53:20)
position=0, limit=2, capacity=2
index_0=1024, index_1=2048
0 0 4 0 0 0 8 0 0 0
```

## 7. ByteBuffer(계속)

### □ 다른 데이터 형식으로 읽고 쓰기

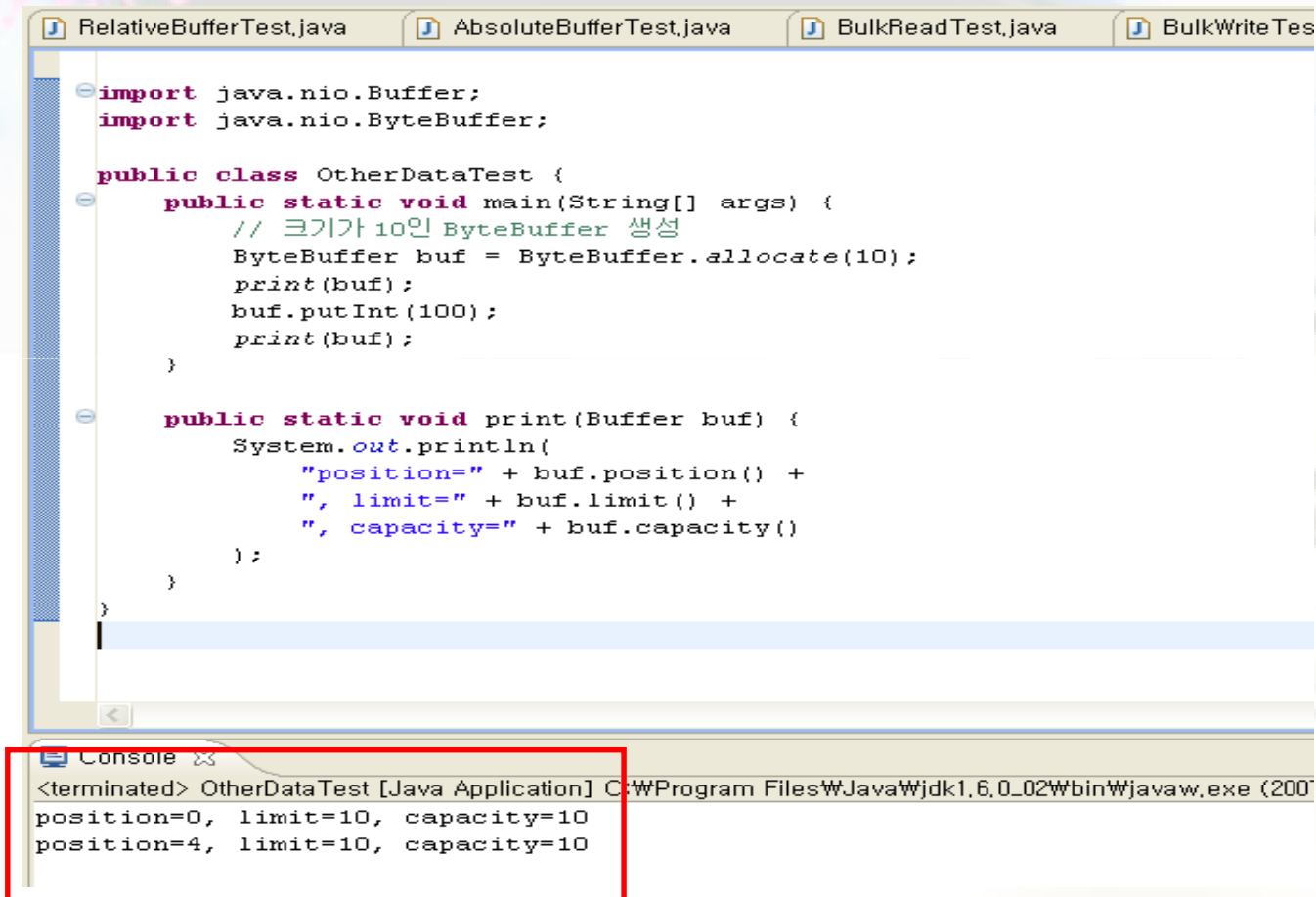
```
public abstract class ByteBuffer
    extends Buffer
    implements Comparable {

    public abstract char getChar();
    public abstract char getChar(int index);
    public abstract short getShort();
    public abstract short getShort(int index);
    public abstract int getInt();
    public abstract int getInt(int index);
    public abstract long getLong();
    public abstract long getLong(int index);
    public abstract float getFloat();
    public abstract float getFloat(int index);
    public abstract double getDouble();
    public abstract double getDouble(int index);

    public abstract ByteBuffer putChar(char value);
    public abstract ByteBuffer putChar(int index, char value);
    public abstract ByteBuffer putShort(short value);
    public abstract ByteBuffer putShort(int index, short value);
    public abstract ByteBuffer putInt(int value);
    public abstract ByteBuffer putInt(int index, int value);
    public abstract ByteBuffer putLong(long value);
    public abstract ByteBuffer putLong(int index, long value);
    public abstract ByteBuffer putFloat(float value);
    public abstract ByteBuffer putFloat(int index, float value);
    public abstract ByteBuffer putDouble(double value);
    public abstract ByteBuffer putDouble(int index, double value);
}
```

## 7. ByteBuffer(계속)

### □ 다른 데이터 형식으로 읽고 쓰기(예제)



```
RelativeBufferTest.java AbsoluteBufferTest.java BulkReadTest.java BulkWriteTest.java

import java.nio.Buffer;
import java.nio.ByteBuffer;

public class OtherDataTest {
    public static void main(String[] args) {
        // 크기가 10인 ByteBuffer 생성
        ByteBuffer buf = ByteBuffer.allocate(10);
        print(buf);
        buf.putInt(100);
        print(buf);
    }

    public static void print(Buffer buf) {
        System.out.println(
            "position=" + buf.position() +
            ", limit=" + buf.limit() +
            ", capacity=" + buf.capacity()
        );
    }
}
```

Console

```
<terminated> OtherDataTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (200
position=0, limit=10, capacity=10
position=4, limit=10, capacity=10
```

## 8. CharBuffer

### □ CharBuffer에 대해서

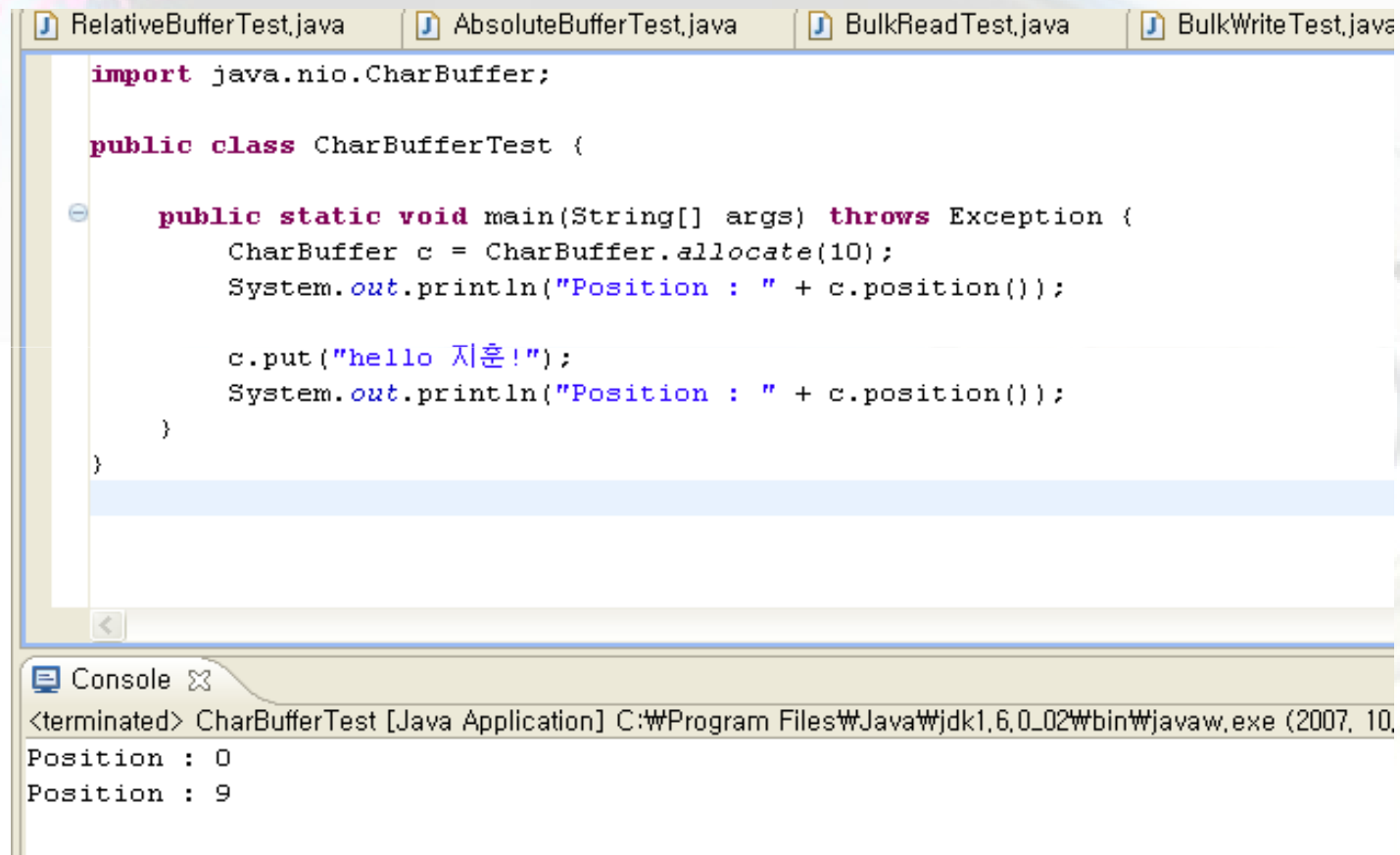
- String 을 이용해서 읽기 전용 뷰 버퍼를 만들 수 있는 메소드 제공
- String 을 버퍼에 쉽게 읽고 쓸수 있는 기능 제공
- String 을 버퍼에 쉽게 쓸 수 있도록 제공해주는 메소드

```
public final CharBuffer put (String src);
```

```
public CharBuffer pub (String src, int offset, int length);
```

## 8. CharBuffer(계속)

### □ CharBuffer에 대해서(예제)



The screenshot shows a Java IDE with four tabs: RelativeBufferTest.java, AbsoluteBufferTest.java, BulkReadTest.java, and BulkWriteTest.java. The active tab is RelativeBufferTest.java, which contains the following code:

```
import java.nio.CharBuffer;

public class CharBufferTest {

    public static void main(String[] args) throws Exception {
        CharBuffer c = CharBuffer.allocate(10);
        System.out.println("Position : " + c.position());

        c.put("hello 지훈!");
        System.out.println("Position : " + c.position());
    }
}
```

Below the code editor is a console window titled "Console" with the following output:

```
<terminated> CharBufferTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007, 10,
Position : 0
Position : 9
```



## 8. CharBuffer(계속)

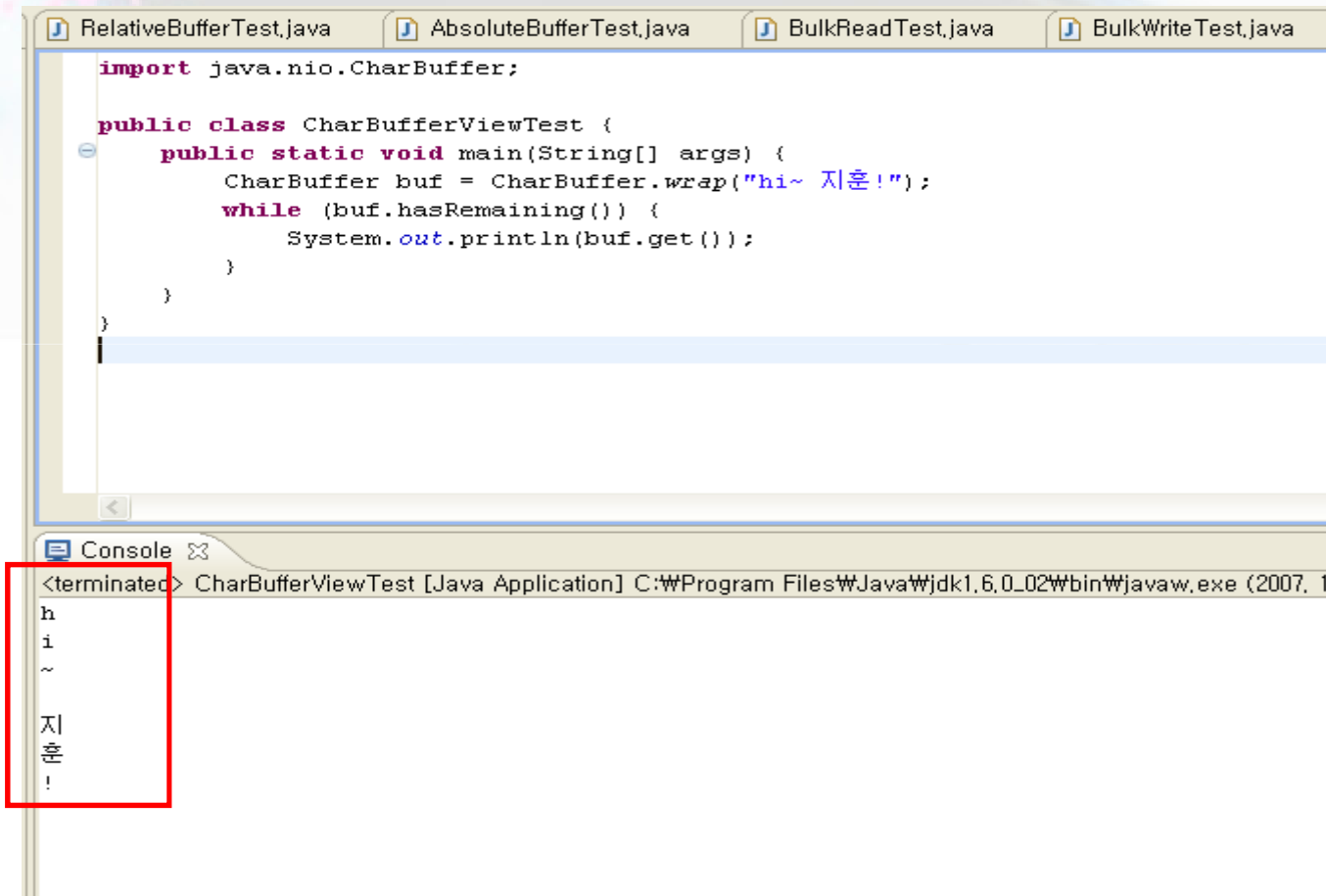
### □ wrap() 사용한 읽기 전용 버퍼 생성

```
public abstract class CharBuffer extends Buffer implements CharSequence, Comparable{  
  
    public static CharBuffer wrap(CharSequence csq);  
    public static CharBuffer wrap(CharSequence esq, int start, int end);  
  
}
```

- 기존의 배열을 이용한 wrap() 메소드와 유사
- CharSequence 인터페이스 (J2SDK 1.4 에서 추가된 인터페이스)는 정규표현식의 도움을 받기 위해 추가
- CharSequence를 구현한 클래스로는 String, StringBuffer, CharBuffer
- CharBuffer는 String또는 StringBuffer를 가지고 읽기 전용 버퍼를 쉽게 만들 수 있는 wrap()메소드 제공

## 8. CharBuffer(계속)

### □ wrap() 사용한 읽기 전용 버퍼 생성(예제)



The screenshot shows a Java IDE with four tabs: RelativeBufferTest.java, AbsoluteBufferTest.java, BulkReadTest.java, and BulkWriteTest.java. The active tab is RelativeBufferTest.java, which contains the following code:

```
import java.nio.CharBuffer;

public class CharBufferViewTest {
    public static void main(String[] args) {
        CharBuffer buf = CharBuffer.wrap("hi~ 지훈!");
        while (buf.hasRemaining()) {
            System.out.println(buf.get());
        }
    }
}
```

Below the code editor is a console window titled "Console". It shows the output of the program, which is the string "hi~ 지훈!" printed character by character. The output is enclosed in a red box:

```
<terminated> CharBufferViewTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007, 1
h
i
~
지
훈
!
```

## 8. CharBuffer(계속)

```
public class CharBufferViewTest2
{
    public static void main(String[] args)
    {
        ByteBuffer buf = ByteBuffer.allocate(100);
        //뷰 버퍼를 생성한다.
        CharBuffer cbuf = buf.asCharBuffer();

        cbuf.put("Hello World!");
        cbuf.flip();

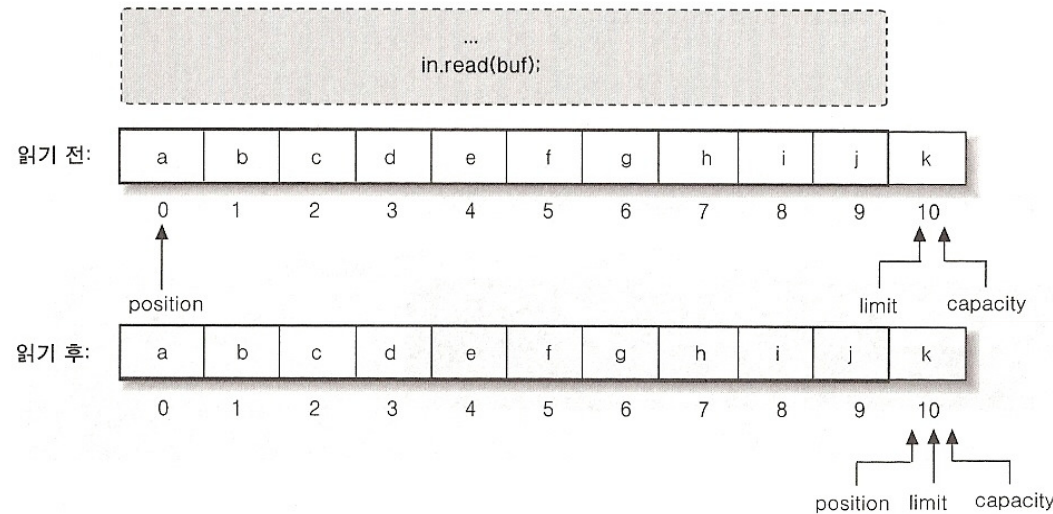
        //toString 메소드로 버퍼 안의 데이터를 스트링으로 만든다.
        //toString 메소드는 포지션에 영향을 주지 않는다.
        String s = cbuf.toString();
        System.out.println("Data: " + s);
        System.out.println("Buffer Position: " + cbuf.position());

        int start = 6;
        int end = 12;
        //버퍼안의 데이터 일부만을 CharSequence로 가져온다.
        CharSequence sub = cbuf.subSequence(start, end);
        System.out.println(sub.toString());
    }
}
```

```
Data: Hello World!
Buffer Position: 0
World!
```

## 9. 채널에서 버퍼 사용하기

### □ 채널의 데이터를 버퍼로 읽어 들이기

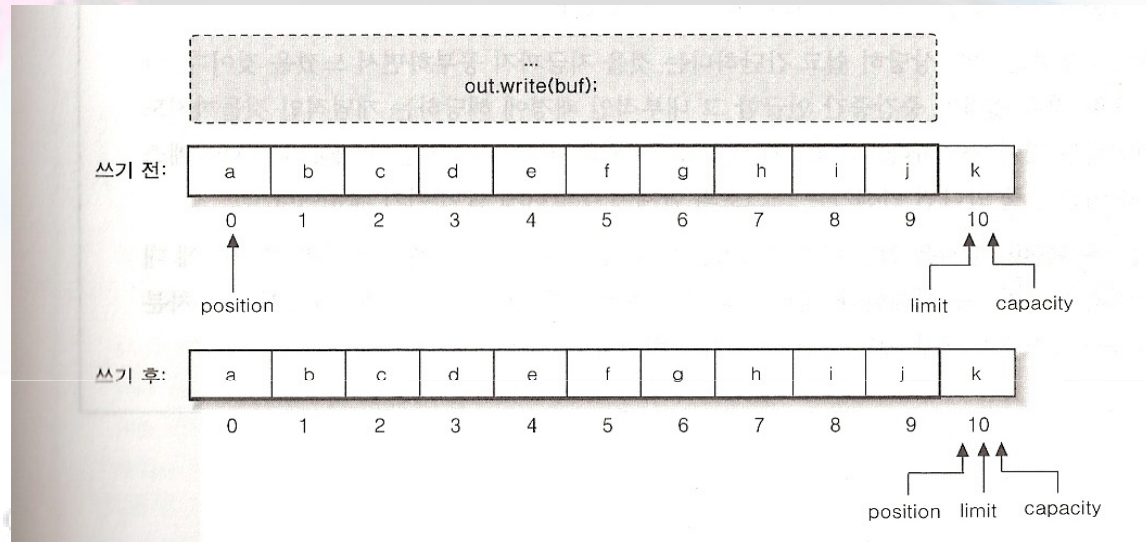


[그림 13-30] 채널에서 버퍼로 데이터 읽기

- 상대적 위치를 이용해서 버퍼에 데이터를 읽어 들이던 메소드들처럼 position 이 limit까지 이동되며 데이터를 읽어들이는다.

## 9. 채널에서 버퍼 사용하기(계속)

### □ 버퍼에 저장된 데이터를 채널에 쓰기



- position에서 limit까지의 데이터를 채널로 쓰고 position도 맞춰서 이동