

Chapter 14. 채널

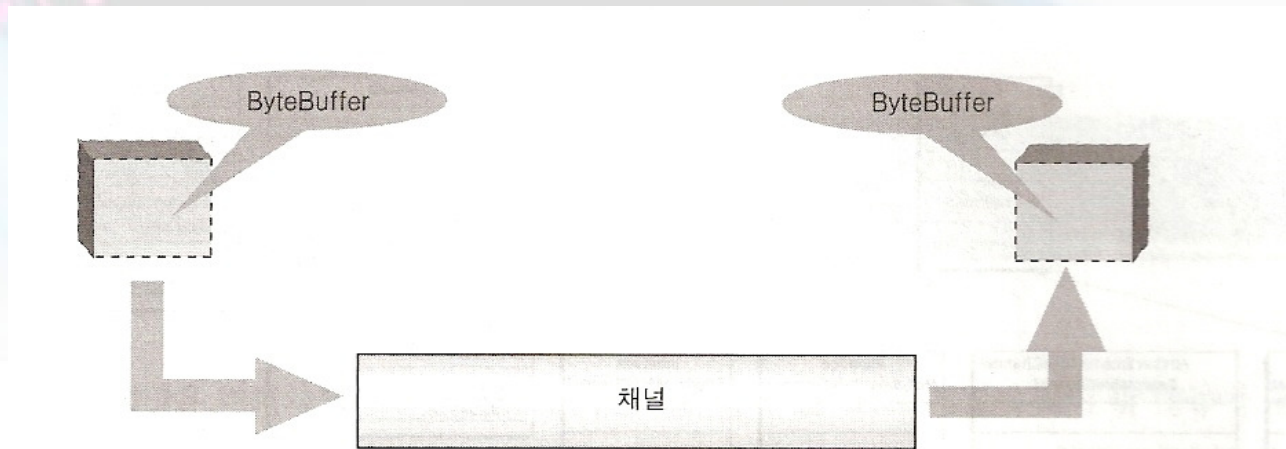
Originally made by Prof. Hanku Lee

Modified by Mingyu Lim

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

1. 채널 개요

□ 채널의 동작 원리



- 데이터를 전달하는 것은 스트림과 유사
(채널이 스트림의 확장이나 발전된 형태는 아님)
- 일종의 게이트웨이
- 파일이나 소켓등에서 사용하던 스트림이 네이티브 IO 서비스를 이용하도록 메소드 제공

1. 채널 개요 (계속)

□ 채널과 스트림의 차이점

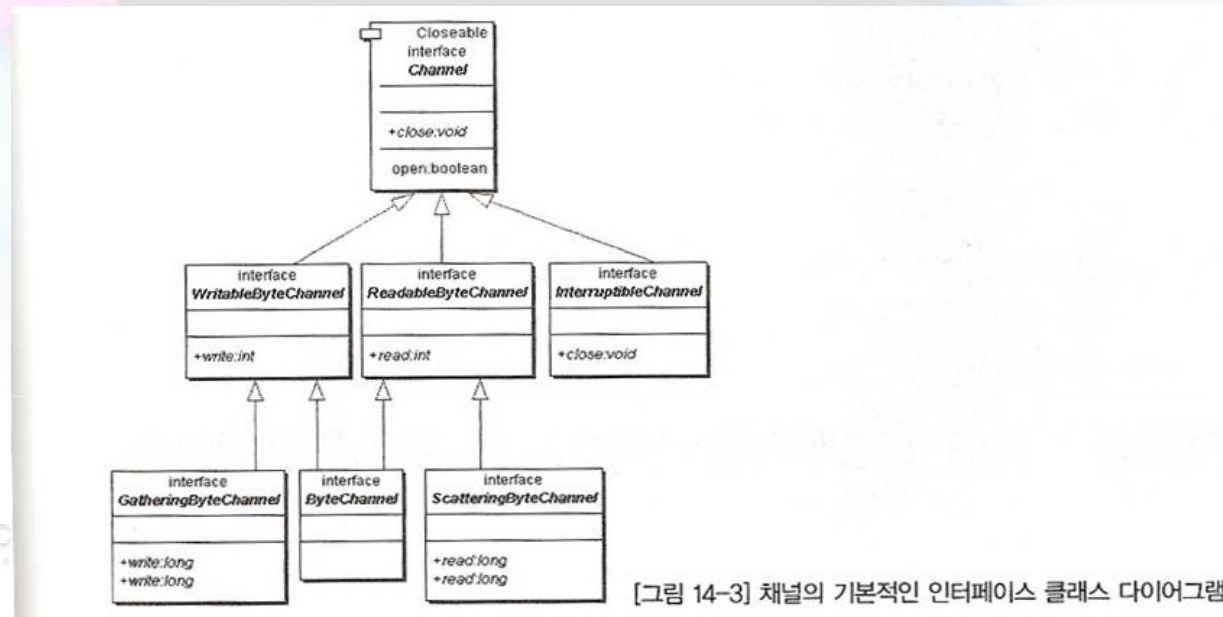
- 데이터를 보내거나 받기 위한 타겟으로 ByteBuffer 를 사용
- 채널을 통해 데이터를 주고 받으면, 운영체제 수준의 네이티브 IO 서비스를 직간접적으로 이용
(메모리 맵 파일, 파일 락킹 같은 기능 사용가능)
- 채널은 스트림과 달리 단방향뿐만 아니라 양방향 통신 가능
(소켓 채널 : 양방향 가능, 파일 채널은 불가능)

❑ java.nio.channel 패키지



2. 채널의 기본 인터페이스

□ 채널의 기본 인터페이스



- 채널은 인터페이스 기반 (행동에 대한 정의만 하고 세부적인 구현은 각각의 하위 클래스에서 담당)
- 각각의 운영체제마다 IO에 관련된 시스템 콜 명령어와 처리루틴이 다르기 때문에 인터페이스로 구현

2. 채널의 기본 인터페이스(계속)

□ Channel, InterruptibleChannel

- Channel 인터페이스 API

```
package java.nio.channel

import java.io.Closeable;
import java.io.IOException;

public interface Channel extends Closeable{
    public boolean isOpen();
    public void close() throws IOException;
}
```

: 단지 채널이 열려있는지의 여부를 확인하기위한 isOpen()메소드와
열려있을경우 채널을 닫기 위한 close() 메소드만 정의

2. 채널의 기본 인터페이스(계속)

□ Channel, InterruptibleChannel

- InterruptibleChannel

```
package java.nio.channel  
  
import java.io.IOException;  
  
public interface InterruptibleChannel extends Channel{  
    public void close() throws IOException;  
}
```

- : 비동기적으로 채널을 닫거나(close) 인터럽트 할수 있는 인터페이스
- : Channel 인터페이스를 상속받고 Channel 인터페이스의 close() 메소드 오버라이딩 한다.

2. 채널의 기본 인터페이스(계속)

□ Channel, InterruptibleChannel

- 기존 스트림에서는 스레드가 블록되거나 어떤 예외상황의 발생으로 인해 비정상적으로 종료되는 경우, 스트림은 여전히 열려 있어 상태 불일치가 종종 발생
- 비동기적 채널 종료
 - ◆ 블록된 스레드가 있는 경우, 다른 스레드가 이 채널의 `close()` 메소드를 호출하여 채널 종료 가능. 블록되었던 스레드는 `AsynchronousCloseException` 수신
 - ◆ 다른 스레드가 블록된 스레드의 `interrupt()` 메소드 호출하는 경우, 채널이 닫히고 블록된 스레드는 `ClosedByInterruptException` 수신

2. 채널의 기본 인터페이스(계속)

□ ReadableByteChannel, WritableByteChannel, ByteChannel

- ReadableByteChannel

```
package java.nio.channels

import java.io.IOException;
import java.nio.ByteBuffer;

public interface ReadableByteChannel extends Channel{
    public int read (ByteBuffer dst) throws IOException
}
```

: 파라미터로 주어진 ByteBuffer로 채널안의 데이터를 읽어 들이는
read() 메소드만 제공

2. 채널의 기본 인터페이스(계속)

□ ReadableByteChannel, WritableByteChannel, ByteChannel

- WritableByteChannel

```
package java.nio.channels

import java.io.IOException;
import java.nio.ByteBuffer;

public interface WritableByteChannel extends Channel{
    public int write(ByteBuffer src) throws IOException
}
```

: ByteBuffer에 저장된 데이터를 채널로 쓰기위한 write() 메소드만 제공

ACHROMATIC COLOR

2. 채널의 기본 인터페이스(계속)

□ ReadableByteChannel, WritableByteChannel, ByteChannel

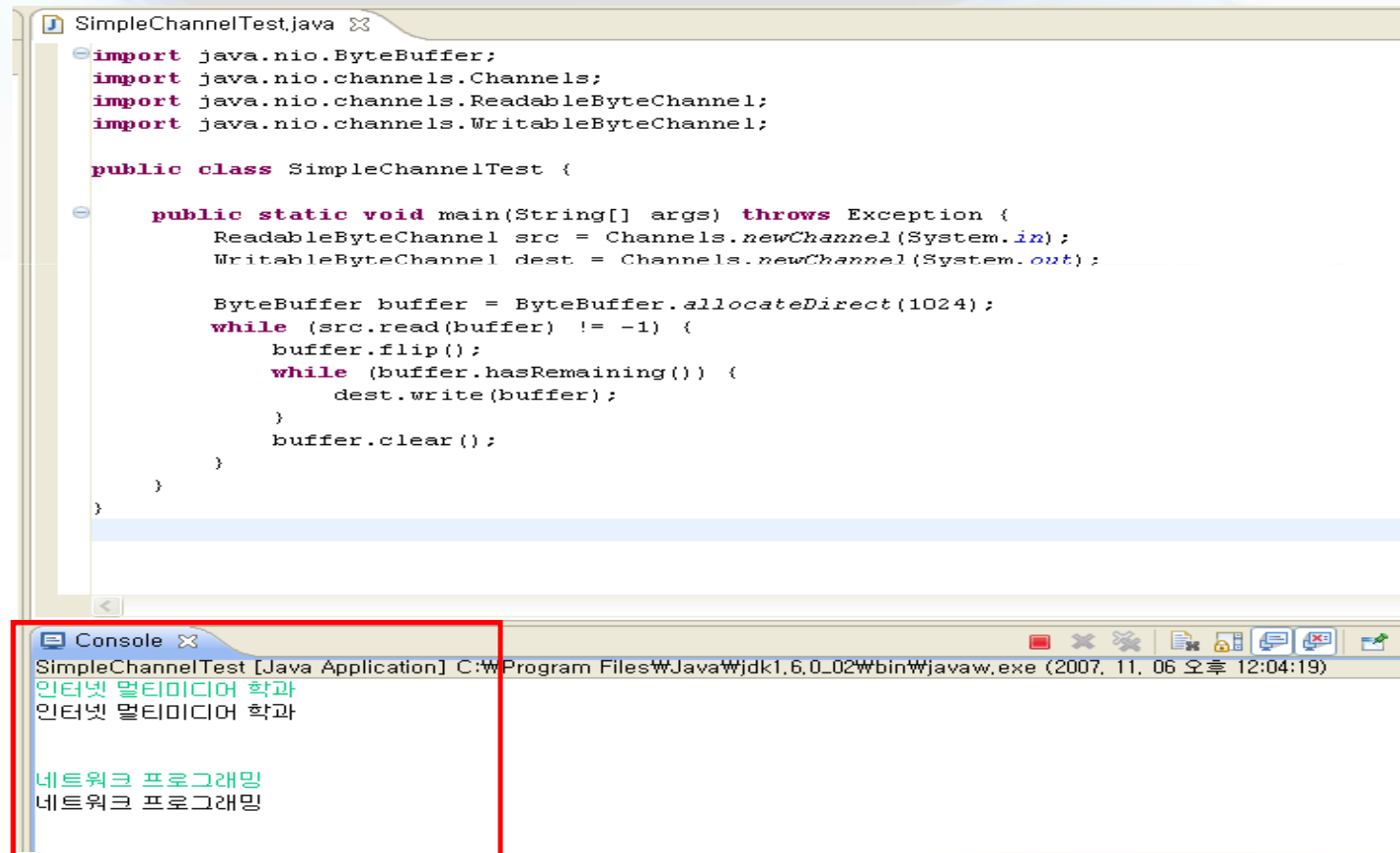
- ByteChannel

: ReadableByteChannel과 WritableByteChannel을 상속해서 채널로부터 데이터를 읽어 들이거나 채널로 데이터를 쓰는 두가지 동작을 모두 할수 있는 인터페이스

2. 채널의 기본 인터페이스(계속)

❑ ReadableByteChannel, WritableByteChannel, ByteChannel

-SimpleChannelTest 예제



```
SimpleChannelTest.java
import java.nio.ByteBuffer;
import java.nio.channels.Channels;
import java.nio.channels.ReadableByteChannel;
import java.nio.channels.WritableByteChannel;

public class SimpleChannelTest {

    public static void main(String[] args) throws Exception {
        ReadableByteChannel src = Channels.newChannel(System.in);
        WritableByteChannel dest = Channels.newChannel(System.out);

        ByteBuffer buffer = ByteBuffer.allocateDirect(1024);
        while (src.read(buffer) != -1) {
            buffer.flip();
            while (buffer.hasRemaining()) {
                dest.write(buffer);
            }
            buffer.clear();
        }
    }
}
```

Console

SimpleChannelTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 11. 06 오후 12:04:19)

인터넷 멀티미디어 학과

인터넷 멀티미디어 학과

네트워크 프로그래밍

네트워크 프로그래밍

2. 채널의 기본 인터페이스(계속)

□ ScatteringByteChannel, GatheringByteChannel

- API

```
public interface ScatteringByteChannel extends ReadableByteChannel{
    public long read(ByteBuffer [] dsts) throws IOException
    public long read(ByteBuffer [] dsts, int offset, int length) throws
    IOException
}
```

```
public interface GatheringByteChannel extends WritableByteChannel{
    public long write(ByteBuffer [] srcs) throws IOException
    public long write(ByteBuffer [] srcs, int offset, int length) throws
    IOException
}
```

ACHROMATIC COLOR

2. 채널의 기본 인터페이스(계속)

❑ ScatteringByteChannel, GatheringByteChannel

- 반복문을 제거해서 사용하기 편리하게 만들기 위한 목적이 아니다.
- 시스템콜과 커널영역에서 프로세스 영역으로의 버퍼복사를 줄이거나 없애줌

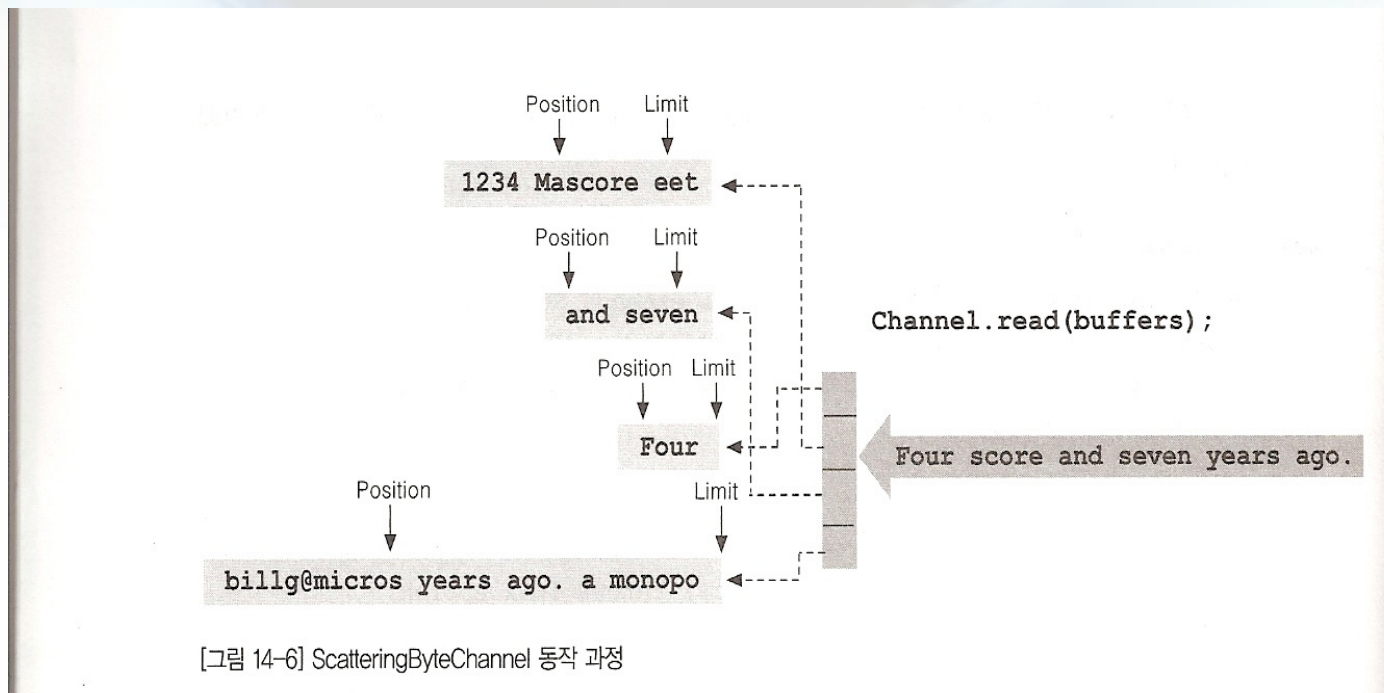
```
ByteBuffer [] buf = ...;
for (int i=0; i<buf.length ; i++ {
    // ByteBuffer 배열의 사이즈만큼 읽기 시스템 콜을 수행
    // 만약 커널과 프로세스 간의 동일한 물리적 메모리 참조하기 않는다면
    // 반복문을 실행하는 횟수만큼 커널 영역에서 프로세스 영역으로
    // 버퍼 복사가 이루어 진다.
    readableByteChannel.read(buf[i])

ByteBuffer [] buf = ...;
    // 단 한번의 시스템 콜을 수행한다.
    // 만약, 앞선 경우와 같이 동일한 물리적 메모리를 참조하지 않는 경우
    // 단 한번의 커널 영역에서 프로세스 영역으로의 버퍼 복사가 이뤄짐
    scatteringByteChannel.read(buf)
```

2. 채널의 기본 인터페이스(계속)

❑ ScatteringByteChannel, GatheringByteChannel

- ScatteringByteChannel 동작



[그림 14-6] ScatteringByteChannel 동작 과정

2. 채널의 기본 인터페이스(계속)

❑ ScatteringByteChannel 예제

```
import java.io.FileInputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.ScatteringByteChannel;

public class ScatterTest {

    public static void main(String[] args) throws IOException {
        FileInputStream fin = new FileInputStream("C://news.txt");
        ScatteringByteChannel channel = fin.getChannel();

        ByteBuffer header = ByteBuffer.allocateDirect(100);
        ByteBuffer body = ByteBuffer.allocateDirect(200);
        ByteBuffer[] buffers = { header, body };

        int readCount = (int) channel.read(buffers);
        channel.close();
        System.out.println("Read Count : " + readCount);

        System.out.println("\n//-----//\n");

        header.flip();
        body.flip();

        byte[] b = new byte[100];
        header.get(b);
        System.out.println("Header : " + new String(b));

        System.out.println("\n//-----//\n");

        byte[] bb = new byte[200];
        body.get(bb);
        System.out.println("Body : " + new String(bb));
    }
}
```

Console

<terminated> ScatterTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 11. 06 오후 12:57:54)

Read Count : 300

//-----//

Header : import java.io.FileInputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import ja

//-----//

Body : va.nio.channels.ScatteringByteChannel;

public class ScatterTest {

public static void main(String[] args) throws IOException {

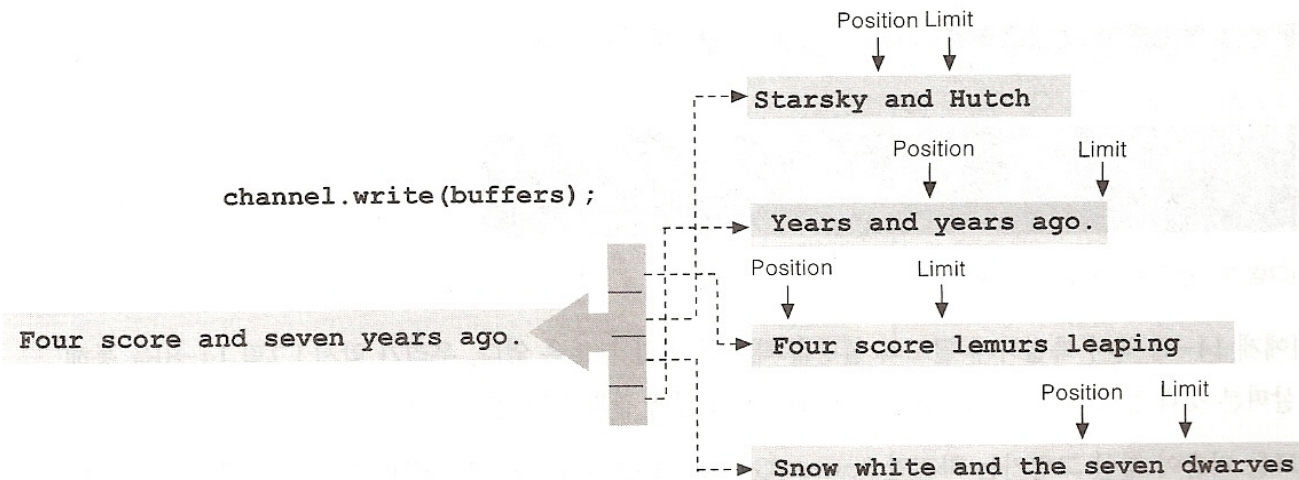
FileInputStream fin = new FileInputStream("C:/news.txt");

Sca

2. 채널의 기본 인터페이스(계속)

❑ ScatteringByteChannel, GatheringByteChannel

- GatheringByteChannel 동작



[그림 14-8] GatheringByteChannel 동작 과정

2. 채널의 기본 인터페이스(계속)

❑ GatheringByteChannel 예제

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.GatheringByteChannel;

public class GatheringTest {

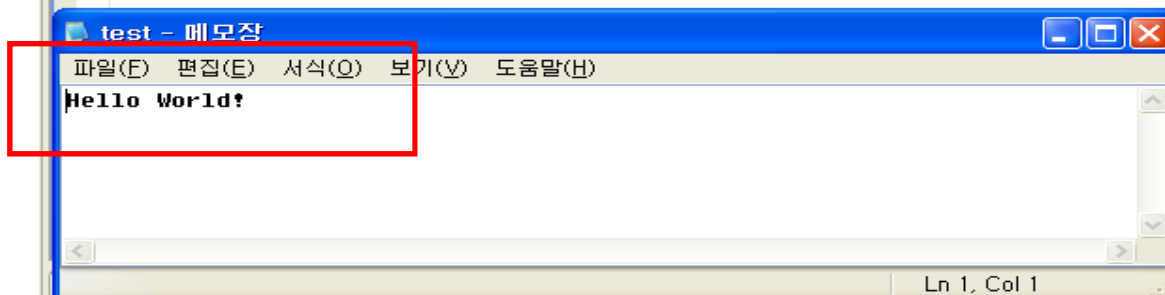
    public static void main(String[] args) throws IOException {
        FileOutputStream fo = new FileOutputStream("C:/test.txt");
        GatheringByteChannel channel = fo.getChannel();

        ByteBuffer header = ByteBuffer.allocateDirect(20);
        ByteBuffer body = ByteBuffer.allocateDirect(40);
        ByteBuffer[] buffers = { header, body };

        header.put("Hello ".getBytes());
        body.put("World!".getBytes());

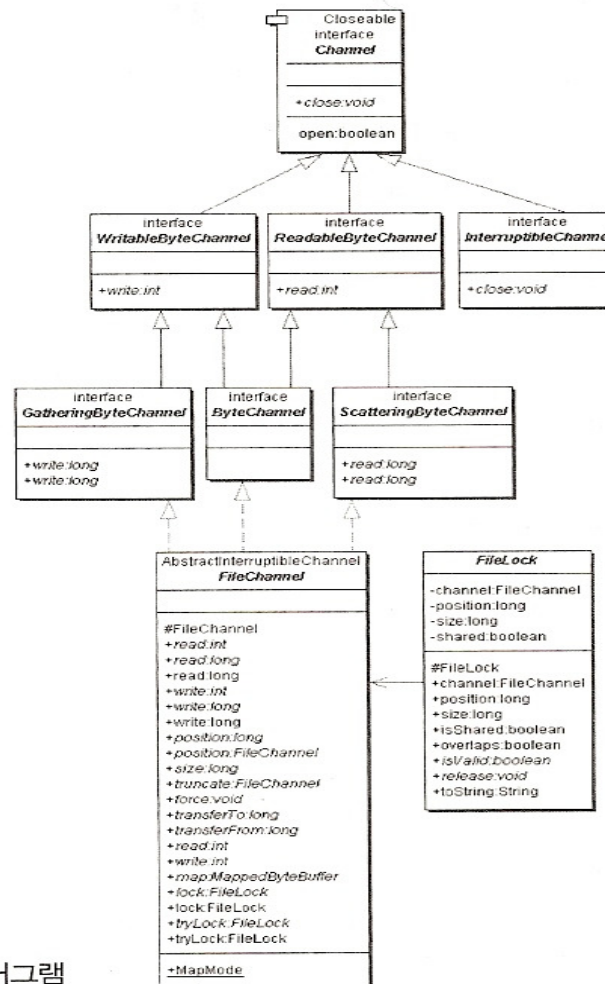
        header.flip();
        body.flip();

        channel.write(buffers);
        channel.close();
    }
}
```



3. 파일 채널

□ 파일채널 클래스 다이어그램



[그림 14-10] 파일채널의 클래스 다이어그램

3. 파일 채널(계속)

□ 파일 채널 인터페이스의 특징

- ByteChannel 인터페이스를 구현
 - : 파일 채널은 읽고 쓸수 있는 양방향성을 가질수 있다.
- AbstractInterruptibleChannel 추상 클래스를 구현
 - : 비동기적인 방식으로 채널을 닫을수 있다
(스레드와 채널 간의 상태불일치가 발생하지 않도록 보장)
- ScatteringByteChannel, GatheringByteChannel 인터페이스를 구현
 - : ByteBuffer 배열을 이용해서 효율적으로 읽기와 쓰기 가능

3. 파일 채널(계속)

□ 파일 채널 일반적인 특징

- 파일 채널은 항상 블로킹 모드며 비블로킹 모드로 설정할수 없다.
- 파일 채널 객체는 직접 만들수 없다.
- 대부분의 채널처럼 파일채널도 가능하면 네이티브 IO 서비스를 사용하기 위해 노력한다.
- 파일 채널 객체는 스레드에 안전하다.

3. 파일 채널(계속)

□ 파일 채널 API

```
package java.nio.channels

public abstract class FileChannel extends AbstractChannel implements
    ByteChannel, GatheringByteChannel, ScatteringByteChannel
{
    // 1. 기본속성
    public abstract int read (ByteBuffer dst)
    public abstract int write(ByteBuffer src)
    public abstract long size()
    public abstract long position()
    public abstract void position (long newPosition)
    public abstract void truncate(long size)
    public abstract void force(boolean metaData)

    // 2. 파일 락킹

    public final FileLock lock()
    public abstract FileLock lock (long position, long size, boolean shared)
    public final FileLock tryLock()
    public abstract FileLock tryLock (long position, long size, boolean
    shared)
```

3. 파일 채널(계속)

□ 파일 채널 API

```
// 3. 메모리 매핑

public abstract MappedByteBuffer map(MapMode mode, long position, long size)
public static class MapMode
{
    public static final MapMode READ_ONLY
    public static final MapMode READ_WRITE
    public static final MapMode PRIVATE
}

// 4. 채널간 직접 전송

public abstract long transferTo (long position, long count, WritableByteChannel
target)

public abstract long transferFrom(
ReadableByteChannel src, long position, long count)
```

3. 파일 채널(계속)

□ 파일 채널

- 기본 속성(1)

- POSIX 시스템 콜에 대응되는 FileChannel, RandomAccessFile 메소드

FileChannel	RandomAccessFile	POSIX system call
read()	read()	read()
write()	write()	write()
size()	length()	fstat()
position()	getFilePointer()	lseek()
position(long newPosition)	seek()	lseek()
truncate()	setLength()	gtruncate()
force()	getFD().Sync()	fsync()

3. 파일 채널(계속)

□ 파일 채널

• 기본 속성(1)

```
public abstract class FileChannel extends AbstractChannel implements
    ByteChannel, GatheringByteChannel, ScatteringByteChannel

    // 1. 기본속성
    public abstract long position()
    public abstract void position (long newPosition)

    public abstract long size()

    public abstract int read (ByteBuffer dst)
    public abstract int read (ByteBuffer dst, long position)
    public abstract int read (ByteBuffer [] dst)
    public abstract int read (ByteBuffer [] dst, int offset, int length)

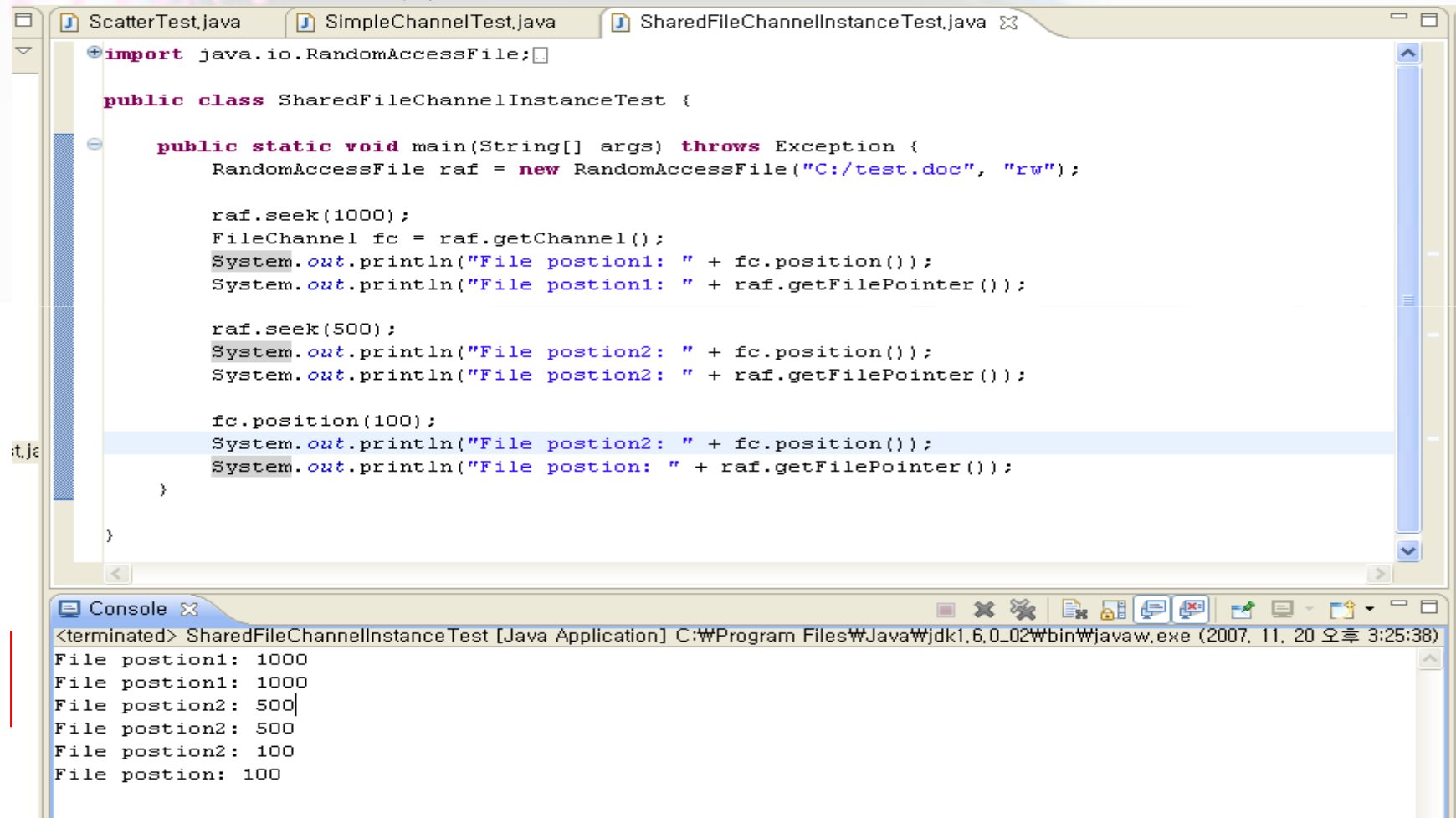
    public abstract int write(ByteBuffer src)
    public abstract int write(ByteBuffer src, long position)
    public abstract int write(ByteBuffer src)
    public abstract int write(ByteBuffer src, int offset, int length)

    public abstract void truncate(long size)
    public abstract void force(boolean metaData)
```

3. 파일 채널(계속)

□ 파일 채널

• 기본 속성 예제(1)



```
import java.io.RandomAccessFile;

public class SharedFileChannelInstanceTest {

    public static void main(String[] args) throws Exception {
        RandomAccessFile raf = new RandomAccessFile("C:/test.doc", "rw");

        raf.seek(1000);
        FileChannel fc = raf.getChannel();
        System.out.println("File postion1: " + fc.position());
        System.out.println("File postion1: " + raf.getFilePointer());

        raf.seek(500);
        System.out.println("File postion2: " + fc.position());
        System.out.println("File postion2: " + raf.getFilePointer());

        fc.position(100);
        System.out.println("File postion2: " + fc.position());
        System.out.println("File postion: " + raf.getFilePointer());
    }
}
```

Console Output:

```
<terminated> SharedFileChannelInstanceTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 11. 20 오후 3:25:38)
File postion1: 1000
File postion1: 1000
File postion2: 500
File postion2: 500
File postion2: 100
File postion: 100
```

3. 파일 채널(계속)

□ 파일 채널

- Public abstract int read(ByteBuffer dst)
- Public abstract int write(ByteBuffer src)
- 버퍼의 읽기, 쓰기과 마찬가지로 자동적으로 파일 포지션이 업데이트
- 파일 끝까지 읽은 경우 EOF를 의미하는 -1 리턴
- 절대적 방식의 읽기, 쓰기 가능
 - ◆ Read()에서 position값이 파일 크기보다 크면, 아무것도 읽지 않고 바로 리턴

□ 버퍼와의 차이

- Write() 메소드로 파일에 데이터를 쓰는 도중에 파일의 끝에 다다른 경우
 - ◆ 버퍼는 Exception 발생
 - ◆ 파일채널은 자동적으로 새롭게 쓰여질 바이트만큼 파일 크기를 늘린다

3. 파일 채널(계속)

□ 파일 채널

- Public abstract void truncate(long size)
 - ◆ 인자로 주어진 크기로 파일 자름
- Public abstract void force(boolean metaData)
 - ◆ 파일의 업데이트된 내용을 강제로 기억장치에 기입
 - ◆ 메타데이터를 함께 업데이트 할 것인지를 인자로 지정

CHROMATIC COLOR
.....

ACHROMATIC COLOR
.....

3. 파일 채널(계속)

□ 파일 채널

- 파일 락킹(2)

1. 파일 락킹은 채널이 아닌 파일을 대상으로 하는것이다.
2. 파일 락킹은 동일한 JVM 내부의 여러스레드 사이가 아닌 외부 프로세스 사이에서 파일의 접근을 제어하기 위한 것이다.

- 파일 채널에서 락을 얻기 위해 제공하는 API

```
public abstract class FileChannel extends AbstractChannel implements
    ByteChannel, GatheringByteChannel, ScatteringByteChannel
{
    public final FileLock lock()
    public abstract FileLock lock (long position, long size, boolean shared)

    public final FileLock tryLock()
    public abstract FileLock tryLock (long position, logn size, boolean
    shared)
}
```

3. 파일 채널(계속)

□ 파일 채널

- 파일 채널에서 락을 얻기 위해 제공하는 API

: lock () 메소드군

1. 락을 얻기위해 우선 시도
2. lock() 파일전체에 대해 배타 락을 획득하는 메소드로 다음을 호출하는 동작을 한다. (즉 파일의 0위치부터 파일이 갖을수 있는 최대 크기 까지의 락을 설정)
lock(0L, Long.MAX_VALUE, false)
3. lock(long position, long size, boolean shared)
첫번째 파라미터는 파일의 락을 설정할 시작 위치
두번째 파라미터는 시작위치로부터 락을 설정할 크기 지정
세번째 파라미터는 true 경우 공유락으로 false 일경우 배타락 설정

3. 파일 채널(계속)

□ 파일 채널

- 파일 채널에서 락을 얻기 위해 제공하는 API

: tryLock() 메소드군

1. 앞의 메소드와 동일한 행동
2. 앞선 메소드는 블로킹메소드였다면 락을 얻기 위해 시도하는 그리고 다른 프로세스에 의해 락이 이미 획득된 상태라면 null을 리턴하는 비블로킹 메소드라는 점

3. 파일 채널(계속)

□ 파일 채널

- 파일 락킹(2)

- FileLock 객체의 API

```
public abstract class FileChannel
{
    public final FileChannel channel()
    public final long position()
    public final long size()
    public final boolean isShared()
    public final boolean overlaps(long position, long size)
    public abstract boolean isValid()
    public abstract void release() throws IOException
}
```


3. 파일 채널(계속)

□ 파일 채널

- 파일 락킹(2)

- FileLock 객체의 API

- : 파일락 객체는 파일의 락된 부분을 캡슐화한 객체이다.
- : channel() 메소드 제공 (자신을 생성한 파일 채널을 리턴)
- : position() 메소드 제공 (파일의 락 시작부분)
- : size() 메소드 제공 (락이 시작되는 지점부터 size 만큼 락을 설정)
- : isShared() 메소드 제공 (리턴값 true 공유락, false 배타락)
- : overlaps() 메소드 제공 (파라미터로 주어진 position부터 size 만큼의 파일 영역에 오버랩할수 있는 락을 얻을수 있는지 없는지를 질의)
- : isValid() 메소드 제공 (해당 파일 락 객체가 유효한지 질의)
- : release() 메소드 제공 (파일 락 객체가 캡슐화하고 있는 파일의 락된 영역의 락을 해제)

3. 파일 채널(계속)

□ 파일 채널

- 중요한 자원 사용시 반드시 다음과 같은 스타일로 코딩

```
FileLock lock = fileChannel.lock() ;
try {
    //어떤처리..
}catch (Exception e) {
    //예외 핸들링..
}finally {
    // 이곳에서 확실하게 중요 자원에 대한 처리를 마무리한다.
    lock.release();
}
```

ACHROMATIC COLOR

3. 파일 채널(계속)

□ 예제 14-6 FileLockTest.java

```
public class FileLockTest
{
    public static void main(String[] args)
    {
        FileChannel channel = null;
        try{
            File file = new File("test.doc");
            channel = new RandomAccessFile(file, "rw").getChannel();

            FileLock lock = channel.lock(0, Long.MAX_VALUE, true);

            try{
                boolean isShared = lock.isShared();
                System.out.println("Is shared lock? :" + isShared);
            }finally{
                //안전하게 락을 해제한다.
                lock.release();
            }
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            if(channel != null)
            {
                try{
                    channel.close();
                }catch(IOException ex){

                }
            }
        }
    }
}
```

Is shared lock? :true

3. 파일 채널(계속)

□ 파일 채널

• 메모리 맵핑(3)

- 파일채널 클래스는 `map()` 메소드 제공
- 메소드의 호출을 통해 열려진 파일과 특별한 형태의 `ByteBuffer` 사이에 가상 메모리가 생성
- 생성된 가상 메모리는 파일을 저장소로 사용
- 가상 메모리 영역은 이 메소드 호출로 리턴되는 `MappedByteBuffer` 객체가 래핑하게된다.
- 버퍼의 저장소는 메모리가 아닌 파일이다.(버퍼에 어떤 내용을 읽거나 쓰면 바로 파일에 반영)
- 메모리 맵핑 기법을 통한 파일 접근은 효율적(자바 io방법 보다)
 - : 시간을 소비하는 명시적인 시스템 콜이 필요없다.
 - : OS 의 가상메모리 시스템은 자동적으로 메모리 페이지들을 캐시

3. 파일 채널(계속)

□ 파일 채널

- 메모리 맵핑(3)

- 세가지 맵 모드

모드	설명
READ_ONLY	버퍼에서 읽는 것만 가능할 경우, 버퍼를 수정하려고 하면 ReadOnlyBufferException 이 발생한다.
READ_WRITE	버퍼에 읽기와 쓰기가 모두 가능하다.
PRIVATE	Copy-on-Write 방식을 사용하므로 읽기와 쓰기가 모두 가능하지만 쓰기를 하는 경우, 복사본을 만들어서 변화 내용을 별도로 보관하며 원본 파일에는 그 변화 내용이 적용 되지 않는다.

- 읽기모드로 파일의 0부터 99까지의 메모리 매핑

```
mapBuffer = fileChannel.map(FileChannel.MapMode.READ_ONLY, 0, 100);
```

- 파일 전체를 메모리 맵핑

```
mapBuffer = fileChannel.map  
(FileChannel.MapMode.READ_ONLY, 0, FileChannel.size());
```

3. 파일 채널(계속)

□ 파일 채널

- 메모리 맵핑(3)

- 메모리 맵핑시 주의점

- : 외부 프로세스가 해당 파일에 읽기, 쓰기 이외의 동작을 하지 못하게 해야 한다.

- : 메모리 맵핑은 파일락과 달리 매핑을 해제하는 메소드가 없다.

- (파일 맵핑이 한번 이루어지면 그 매핑으로 생성된 MappedByteBuffer가 가비지 컬렉트될 때까지 남아있게 된다.)

- MappedByteBuffer API

```
public abstract class MappedByteBuffer extends ByteBuffer {  
    public final MappedByteBuffer load()  
    public final boolean isLoaded()  
    public final MappedByteBuffer force()  
}
```

3. 파일 채널(계속)

□ 파일 채널

- 메모리 맵핑(3)

- MappedByteBuffer API

- : MappedByteBuffer는 다이렉트 버퍼다

- : ByteBuffer를 상속한다.(ByteBuffer 의 모든 기능을 사용가능)

- : load () 메소드

- 1. 운영체제의 시스템 메모리에 캐시되지 않은 부분을 로드하느라

- 걸리는 지연 시간을 없애고 빠르게 사용하기 위해서 메소드 제공

- 2. 운영체제의 시스템 메모리에 매핑된 파일 데이터를 로드할수 있을

- 만큼만 로드한다.

- : isLoaded() 메소드

- 1. 현재 파일이 물리적 메모리에 로드되었는지의 여부를 질의하는

- 메소드

- 2. 정확성은 보장 안됨

- : force() 메소드

- 1. 물리적 기억장치와의 동기화를 유지하기 위해 기억장치에 강제로

- MappedByteBuffer 의 변경사항을 저장하는 역할

3. 파일 채널(계속)

□ 예제 14-7 ByteBufferPool

- 메모리와 파일 매핑을 통해 생성한 버퍼 두가지 유지
- 평소 메모리 버퍼를 사용하고 모두 사용중일 때는 파일 매핑을 통해 생성한 가상 메모리 버퍼를 사용

```
public ByteBufferPool(int memorySize, int fileSize, File file) throws IOException {  
    if (memorySize > 0)  
        initMemoryBuffer(memorySize);  
  
    if (fileSize > 0)  
        initFileBuffer(fileSize, file);  
}
```

- 생성자에서 인자 세개 필요
 - ◆ 물리적 메모리를 사용하기 위한 다이렉트 ByteBuffer 크기
 - ◆ 가상 메모리에 사용할 파일의 크기, 이 때 사용할 파일

3. 파일 채널(계속)

□예제 14-7 ByteBufferPool (계속)

– 메모리, 파일 버퍼 초기화

```
private void initMemoryBuffer(int size) {  
    int bufferCount = size / MEMORY_BLOCKSIZE;  
    size = bufferCount * MEMORY_BLOCKSIZE;  
    ByteBuffer directBuf = ByteBuffer.allocateDirect(size);  
    divideBuffer(directBuf, MEMORY_BLOCKSIZE, memoryQueue);  
}
```

- 메모리 버퍼 한 개 크기(MEMORY_BLOCKSIZE)를 고려하여 버퍼 총 크기 다시 계산
- 큰 덩어리의 ByteBuffer를 실제 사용할 버퍼의 크기로 잘라서 풀로 만들기 위해 divideBuffer() 호출

3. 파일 채널(계속)

□예제 14-7 ByteBufferPool (계속)

```
private void divideBuffer(ByteBuffer buf, int blockSize, ArrayList list) {  
    int bufferCount = buf.capacity() / blockSize;  
    int position = 0;  
    for (int i = 0; i < bufferCount; i++) {  
        int max = position + blockSize;  
        buf.limit(max);  
        list.add(buf.slice());  
        position = max;  
        buf.position(position);  
    }  
}
```

- ByteBuffer를 blockSize로 잘라 list에 넣는다.
- 버퍼를 자르기 위해 slice() 메소드 사용

3. 파일 채널(계속)

□예제 14-7 ByteBufferPool (계속)

```
private void initFileBuffer(int size, File f) throws IOException {
    int bufferCount = size / FILE_BLOCKSIZE;
    size = bufferCount * FILE_BLOCKSIZE;
    RandomAccessFile file = new RandomAccessFile(f, "rw");
    try {
        file.setLength(size);
        ByteBuffer fileBuffer = file.getChannel().map(FileChannel.MapMode.READ_WRITE, 0L, size);
        divideBuffer(fileBuffer, FILE_BLOCKSIZE, fileQueue);
    } finally {
        file.close();
    }
}
```

- 주어진 파일을 임의접근가능 파일로 생성
- 파일 크기를 생성자에서 지정한 총 파일 크기로 설정
- 임의접근파일의 채널을 얻어 map()으로 MappedByteBuffer 생성
- 나누기 위해 divideBuffer() 호출

3. 파일 채널(계속)

□ 파일채널

- 채널간 직접 전송(4)

```
public abstract class FileChannel extends AbstractChannel implements
    ByteChannel, GatheringByteChannel, ScatteringByteChannel
{

    public abstract long transferTo (long position, long count,
        WritableByteChannel target)

    public abstract long transferFrom(
        ReadableByteChannel src, long position, long count)

}
```

- 버퍼를 거치지 않고 채널 사이에서 다이렉트로 데이터를 전송하는 메소드로 파일채널에만 존재하는 특별한 형태의 메소드
(소켓 채널 간에는 메소드를 사용할수 없다.)

3. 파일 채널(계속)

□ 파일채널

- 채널간 직접 전송(4)

- transferTo() 메소드

- : 파라미터로 주어진 target 채널로 현재 파일채널의 position 부터 count 만큼의 데이터를 전송한다.

- : 전송을 요청한 바이트의 일부 또는 전부가 전송되지 않을수 있다.

- (파일채널의 바이트수가 지정된 position부터 시작되는 count보다 적은 경우, 타겟 채널이 비블록 모드의 소켓으로 출력 버퍼의 비어있는 공간이 전송하려는 바이트의 양보다 적은 경우

- transferFrom() 메소드

- : 파라미터로 주어진 src 채널로 부터 최대 count 만큼의 데이터를 읽어 들여 이 파일 채널에 지정된 position에 기입한다.

- : 데이터의 일부 또는 전부를 전송하지 못할수도 있다.

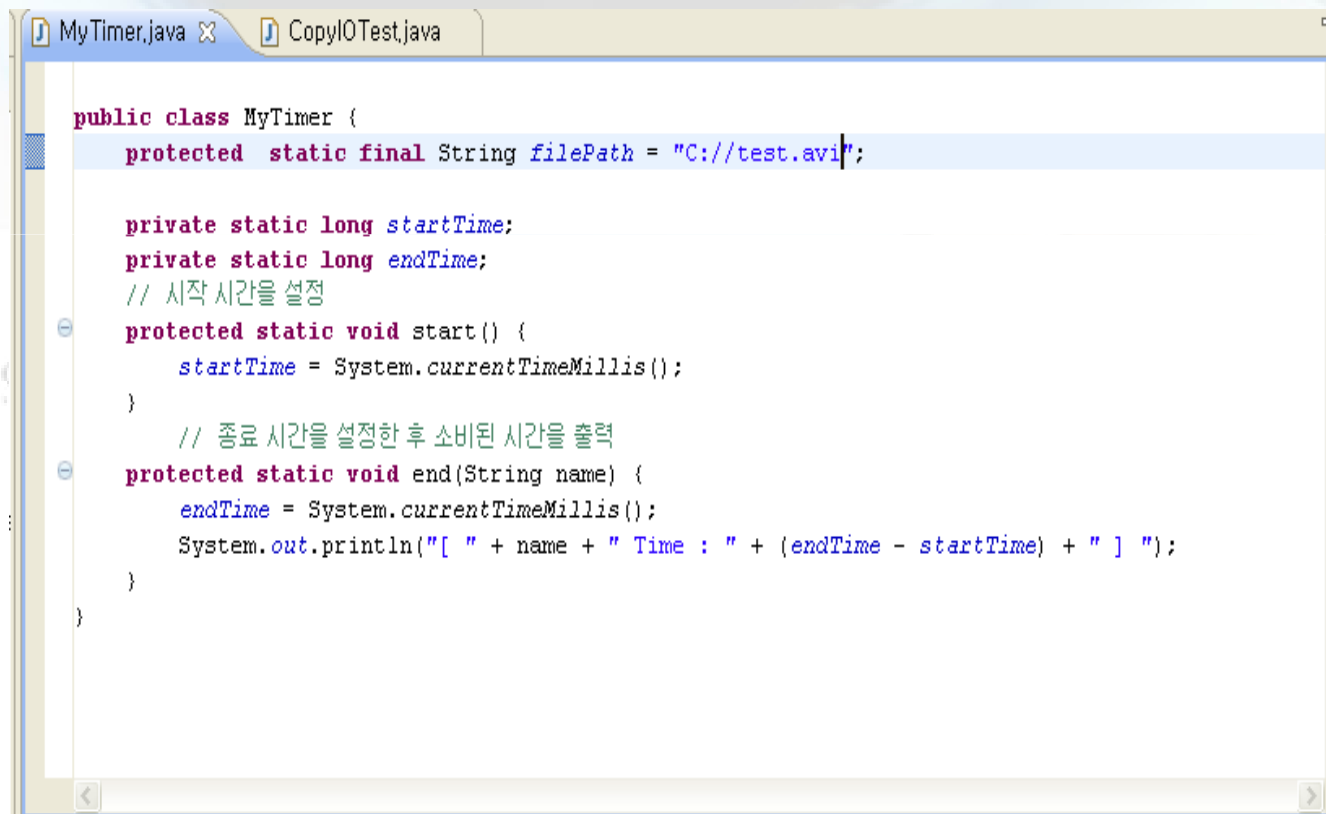
- 파일채널에만 존재

- 파일채널의 포지션을 업데이트하지 않는다

3. 파일 채널(계속)

□ 파일채널

- 파일 복사 프로그램 성능 평가 예제
 - 복사시간 측정을 위한 공통 메소드로 MyTimer 클래스로 분리 생성
(원본파일 test.ppt 크기 : 4.2M)

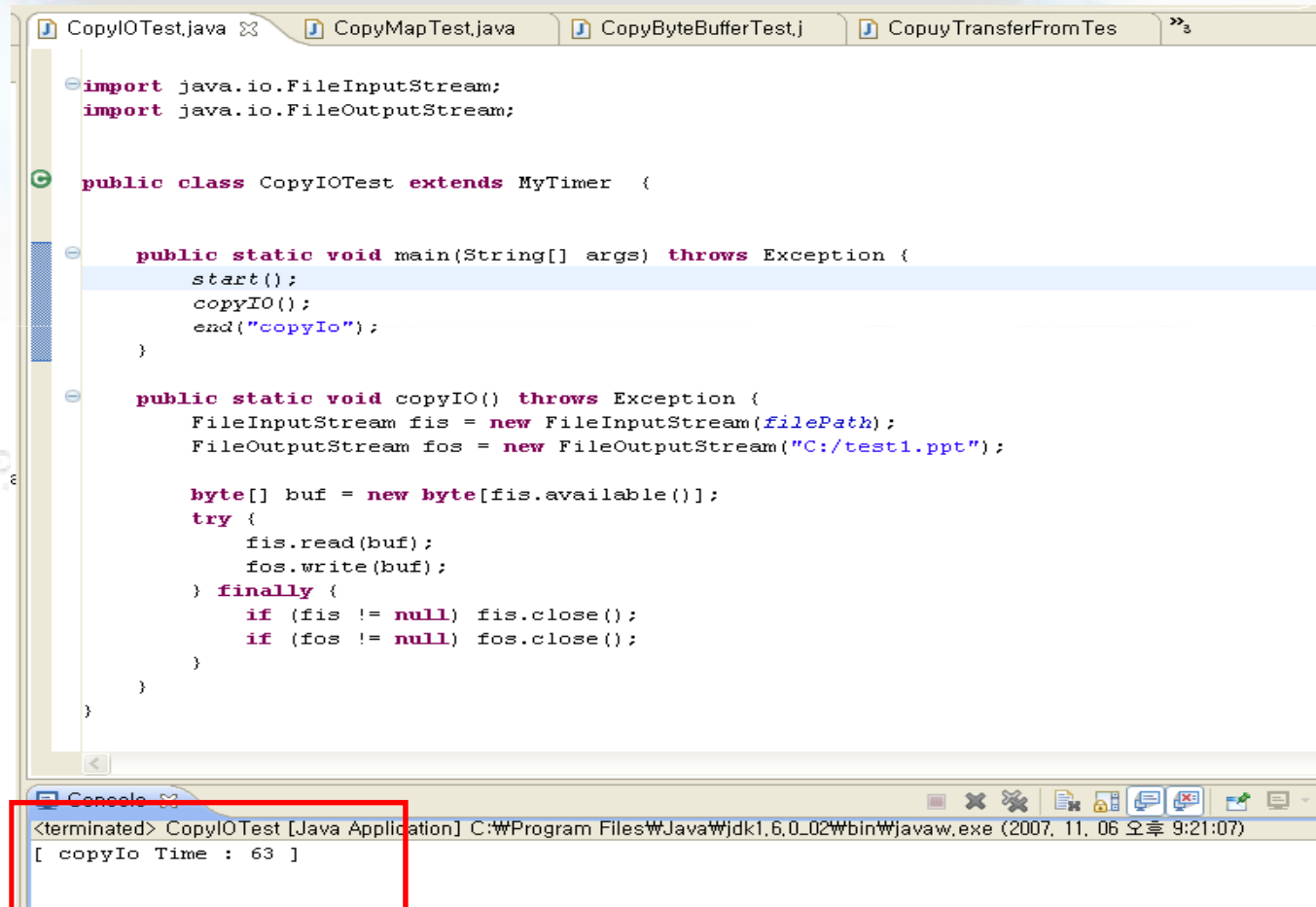


```
public class MyTimer {  
    protected static final String filePath = "C://test.avi";  
  
    private static long startTime;  
    private static long endTime;  
    // 시작 시간을 설정  
    protected static void start() {  
        startTime = System.currentTimeMillis();  
    }  
    // 종료 시간을 설정한 후 소비된 시간을 출력  
    protected static void end(String name) {  
        endTime = System.currentTimeMillis();  
        System.out.println("[ " + name + " Time : " + (endTime - startTime) + " ] ");  
    }  
}
```

3. 파일 채널(계속)

□ 파일채널

- 파일 복사 프로그램 성능 평가 예제(파일의 총크기 만큼 버퍼 생성)



The screenshot shows an IDE with several tabs: CopyIOTest.java, CopyMapTest.java, CopyByteBufferTest.j, and CopuyTransferFromTes. The active tab is CopyIOTest.java, which contains the following Java code:

```
import java.io.FileInputStream;
import java.io.FileOutputStream;

public class CopyIOTest extends MyTimer {

    public static void main(String[] args) throws Exception {
        start();
        copyIO();
        end("copyIo");
    }

    public static void copyIO() throws Exception {
        FileInputStream fis = new FileInputStream(filePath);
        FileOutputStream fos = new FileOutputStream("C:/test1.ppt");

        byte[] buf = new byte[fis.available()];
        try {
            fis.read(buf);
            fos.write(buf);
        } finally {
            if (fis != null) fis.close();
            if (fos != null) fos.close();
        }
    }
}
```

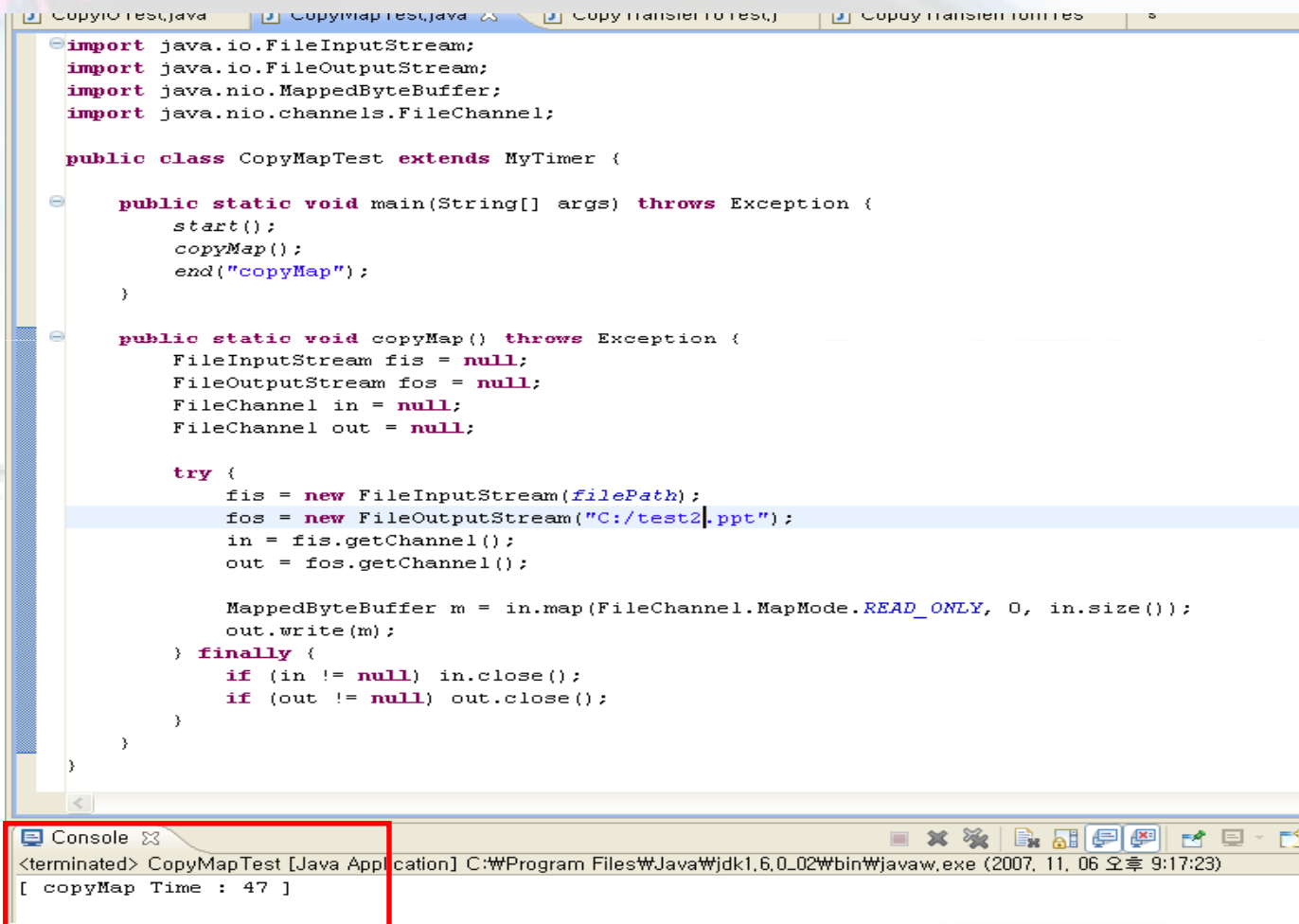
At the bottom, the Console window shows the execution output:

```
<terminated> CopyIOTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 11. 06 오후 9:21:07)
[ copyIo Time : 63 ]
```

3. 파일 채널(계속)

□ 파일채널

- 파일 복사 프로그램 성능 평가 예제(파일 매핑 통한 복사)



```
CopyIO test.java CopyMap test.java CopyTransferIO test.java CopyTransferIO test.java
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;

public class CopyMapTest extends MyTimer {

    public static void main(String[] args) throws Exception {
        start();
        copyMap();
        end("copyMap");
    }

    public static void copyMap() throws Exception {
        FileInputStream fis = null;
        FileOutputStream fos = null;
        FileChannel in = null;
        FileChannel out = null;

        try {
            fis = new FileInputStream(filePath);
            fos = new FileOutputStream("C:/test2.ppt");
            in = fis.getChannel();
            out = fos.getChannel();

            MappedByteBuffer m = in.map(FileChannel.MapMode.READ_ONLY, 0, in.size());
            out.write(m);
        } finally {
            if (in != null) in.close();
            if (out != null) out.close();
        }
    }
}
```

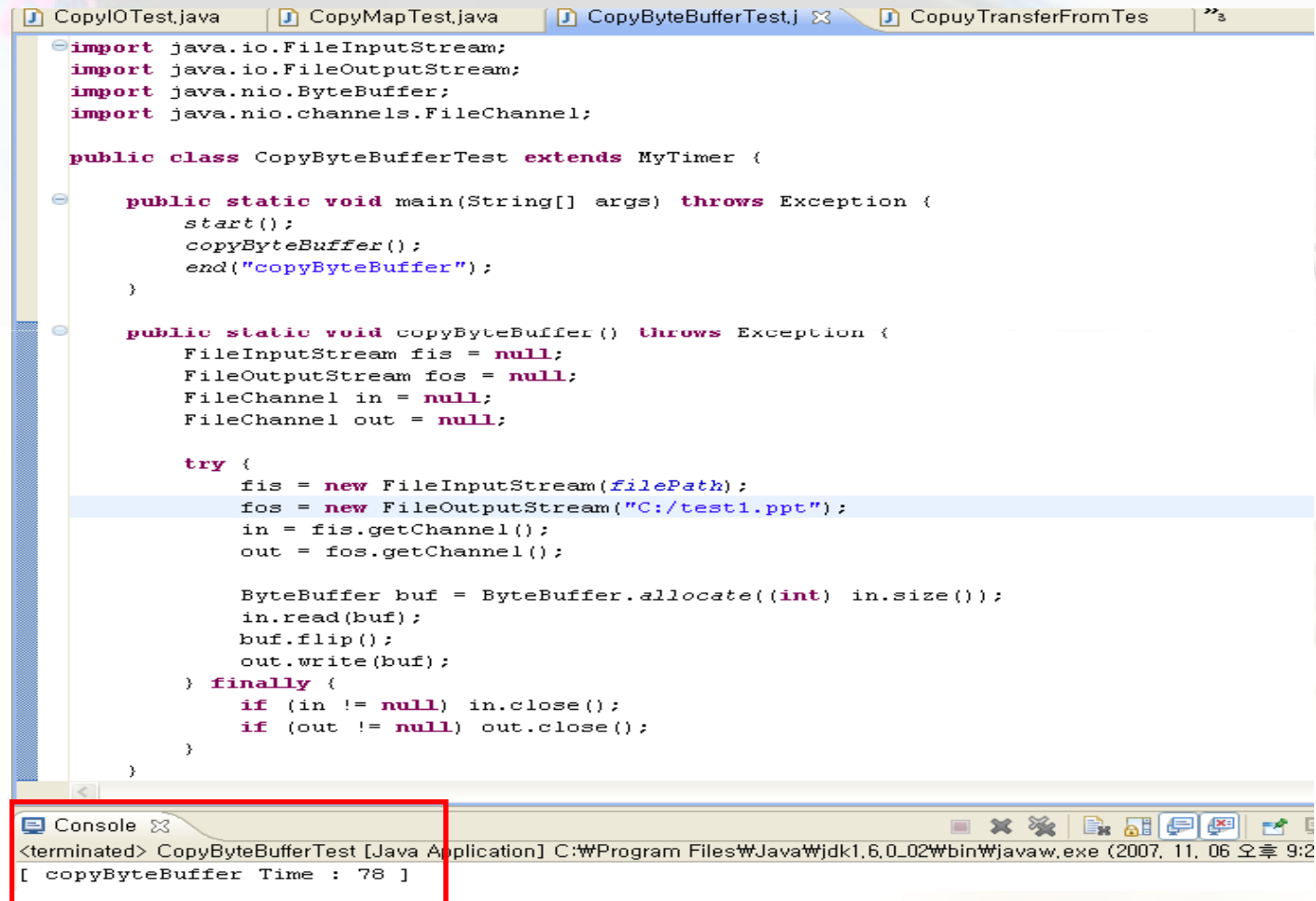
Console

<terminated> CopyMapTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 11. 06 오후 9:17:23)
[copyMap Time : 47]

3. 파일 채널(계속)

□ 파일채널

- 파일 복사 프로그램 성능 평가 예제(ByteBuffer 이용 복사하는 채널사용)



```
CopyIOTest.java CopyMapTest.java CopyByteBufferTest.j CopuyTransferFromTes
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;

public class CopyByteBufferTest extends MyTimer {

    public static void main(String[] args) throws Exception {
        start();
        copyByteBuffer();
        end("copyByteBuffer");
    }

    public static void copyByteBuffer() throws Exception {
        FileInputStream fis = null;
        FileOutputStream fos = null;
        FileChannel in = null;
        FileChannel out = null;

        try {
            fis = new FileInputStream(filePath);
            fos = new FileOutputStream("C:/test1.ppt");
            in = fis.getChannel();
            out = fos.getChannel();

            ByteBuffer buf = ByteBuffer.allocate((int) in.size());
            in.read(buf);
            buf.flip();
            out.write(buf);
        } finally {
            if (in != null) in.close();
            if (out != null) out.close();
        }
    }
}
```

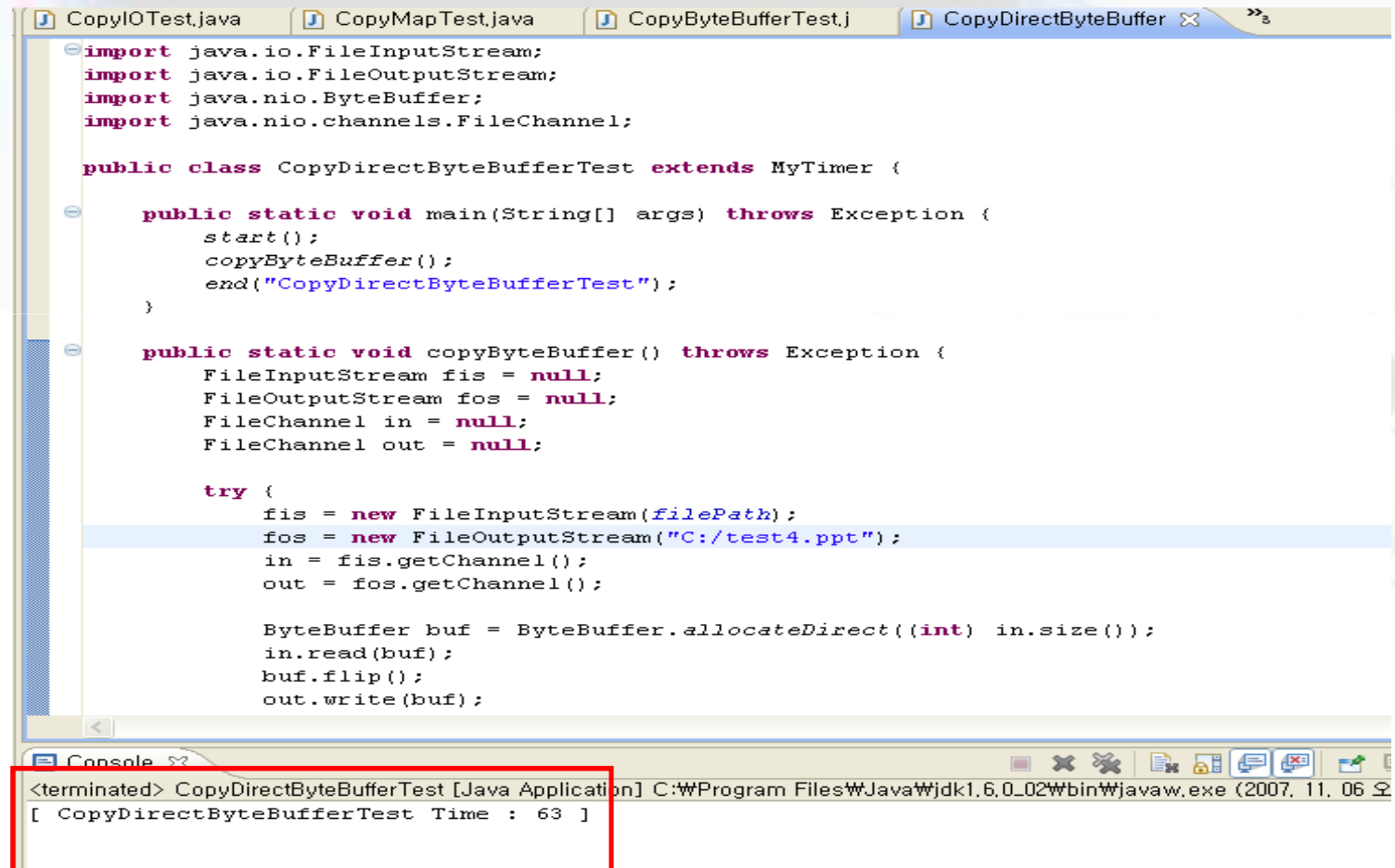
Console

<terminated> CopyByteBufferTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 11. 06 오후 9:20)
[copyByteBuffer Time : 78]

3. 파일 채널(계속)

□ 파일채널

- 파일 복사 프로그램 성능 평가 예제(다이렉트 버퍼 이용)



```
CopyIOTest.java CopyMapTest.java CopyByteBufferTest.j CopyDirectByteBufferTest.java
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;

public class CopyDirectByteBufferTest extends MyTimer {

    public static void main(String[] args) throws Exception {
        start();
        copyByteBuffer();
        end("CopyDirectByteBufferTest");
    }

    public static void copyByteBuffer() throws Exception {
        FileInputStream fis = null;
        FileOutputStream fos = null;
        FileChannel in = null;
        FileChannel out = null;

        try {
            fis = new FileInputStream(filePath);
            fos = new FileOutputStream("C:/test4.ppt");
            in = fis.getChannel();
            out = fos.getChannel();

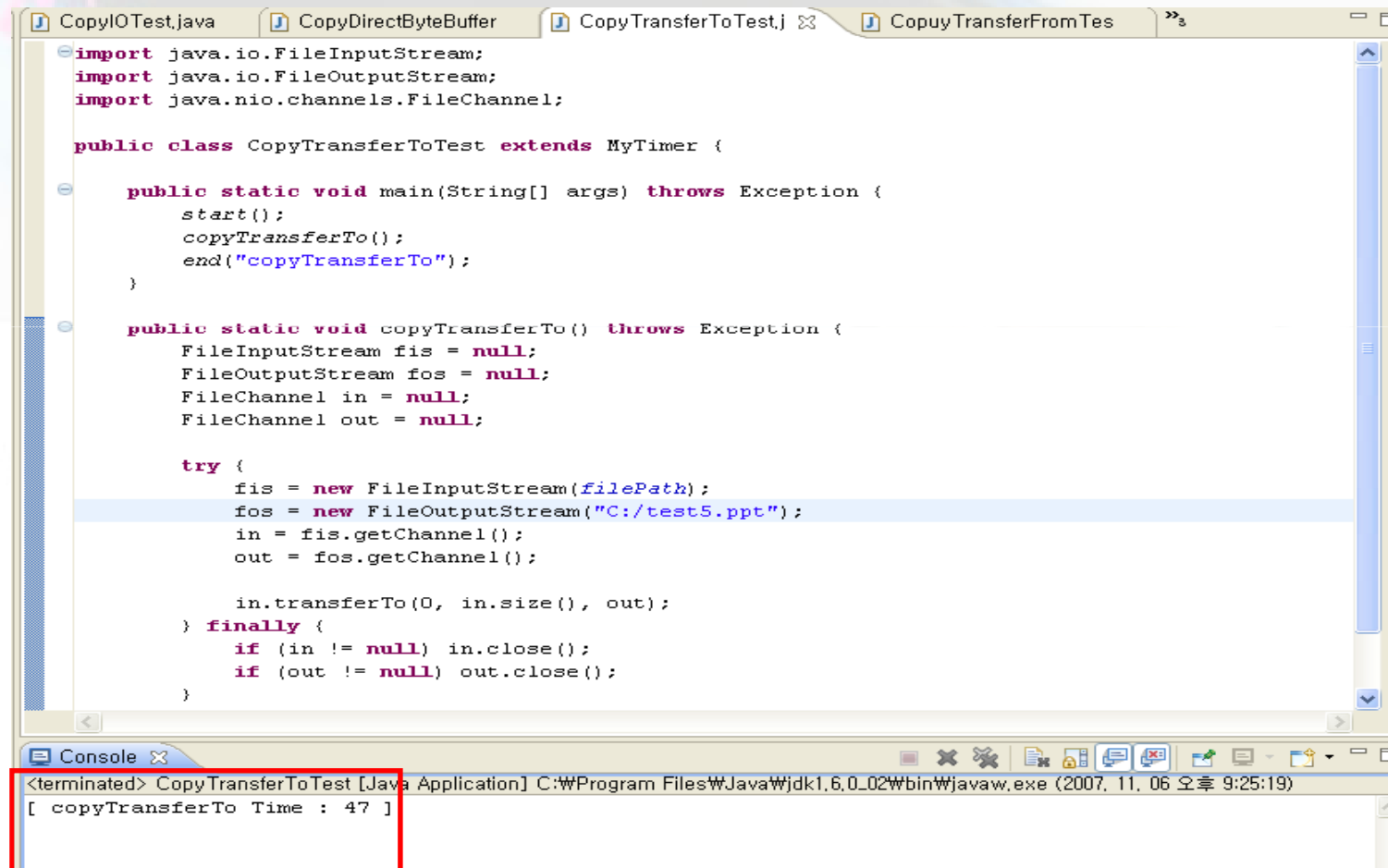
            ByteBuffer buf = ByteBuffer.allocateDirect((int) in.size());
            in.read(buf);
            buf.flip();
            out.write(buf);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

<terminated> CopyDirectByteBufferTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 11. 06 오후 1:06:03)
[CopyDirectByteBufferTest Time : 63]

3. 파일 채널(계속)

□ 파일채널

- 파일 복사 프로그램 성능 평가 예제(채널에서 채널로 복사)



```
CopyIOTest.java CopyDirectByteBuffer CopyTransferToTest.j CopuyTransferFromTes
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.nio.channels.FileChannel;

public class CopyTransferToTest extends MyTimer {

    public static void main(String[] args) throws Exception {
        start();
        copyTransferTo();
        end("copyTransferTo");
    }

    public static void copyTransferTo() throws Exception {
        FileInputStream fis = null;
        FileOutputStream fos = null;
        FileChannel in = null;
        FileChannel out = null;

        try {
            fis = new FileInputStream(filePath);
            fos = new FileOutputStream("C:/test5.ppt");
            in = fis.getChannel();
            out = fos.getChannel();

            in.transferTo(0, in.size(), out);
        } finally {
            if (in != null) in.close();
            if (out != null) out.close();
        }
    }
}
```

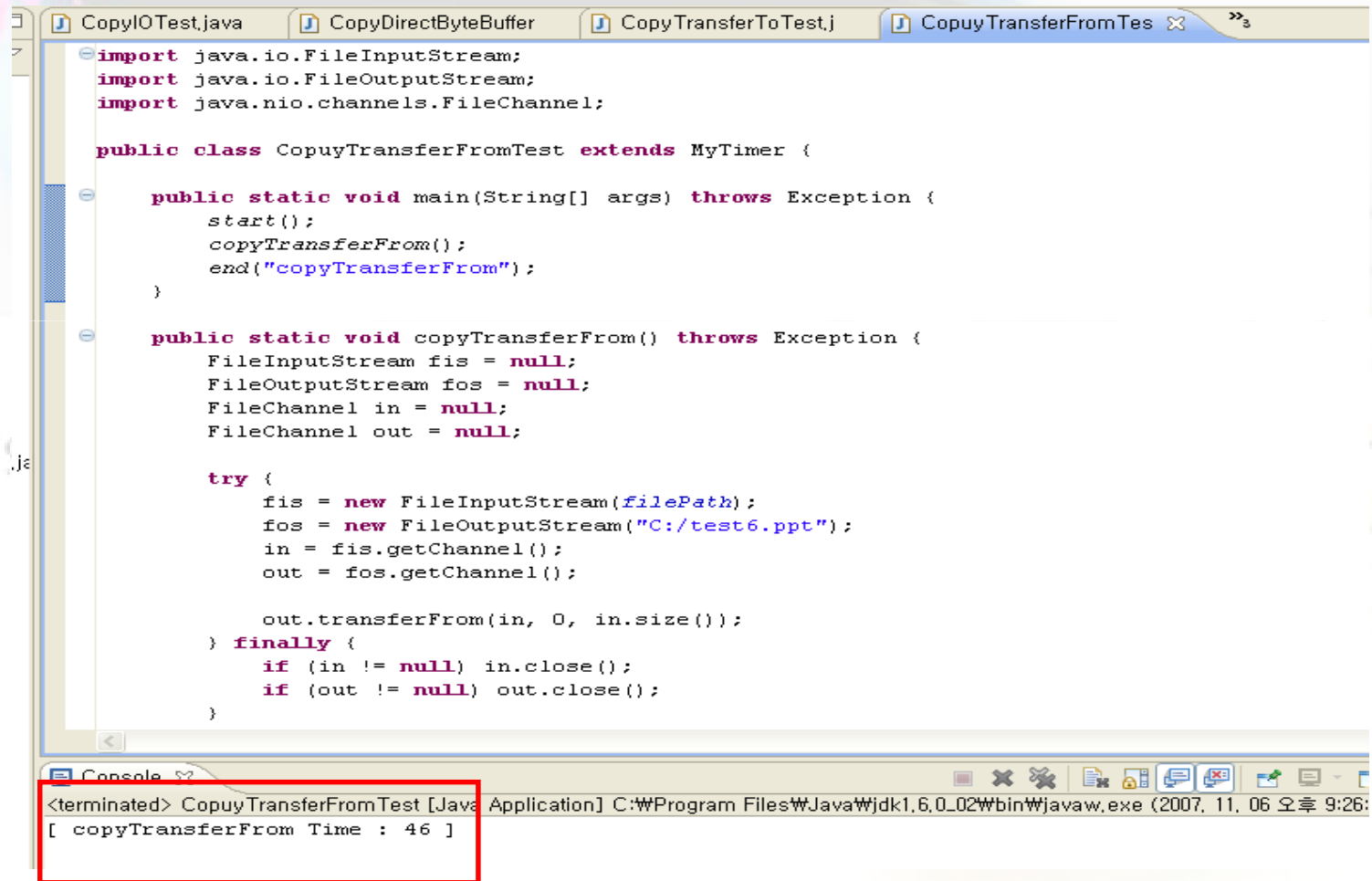
Console

```
<terminated> CopyTransferToTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 11. 06 오후 9:25:19)
[ copyTransferTo Time : 47 ]
```

3. 파일 채널(계속)

□ 파일채널

- 파일 복사 프로그램 성능 평가 예제(채널에서 채널로 복사 : 다른메소드)



```
CopyIOTest.java CopyDirectByteBuffer CopyTransferToTest,j CopuyTransferFromTes x
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.nio.channels.FileChannel;

public class CopuyTransferFromTest extends MyTimer {

    public static void main(String[] args) throws Exception {
        start();
        copyTransferFrom();
        end("copyTransferFrom");
    }

    public static void copyTransferFrom() throws Exception {
        FileInputStream fis = null;
        FileOutputStream fos = null;
        FileChannel in = null;
        FileChannel out = null;

        try {
            fis = new FileInputStream(filePath);
            fos = new FileOutputStream("C:/test6.ppt");
            in = fis.getChannel();
            out = fos.getChannel();

            out.transferFrom(in, 0, in.size());
        } finally {
            if (in != null) in.close();
            if (out != null) out.close();
        }
    }
}
```

<terminated> CopuyTransferFromTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 11. 06 오후 9:26:)
[copyTransferFrom Time : 46]

3. 파일 채널(계속)

□ 파일채널

- 파일 복사 프로그램 성능 평가 예제

복사 테스트 클래스	걸린시간(밀리세컨드)	복사테스트 클래스	걸린시간(밀리세컨드)
CopyIOTest	63	CopyMapTest	47
CopyByteBufferTest	78	CopyDirectByteBufferTest	63
CopyTransferToTest	47	CopyTransferFromTest	46

- 교재와 순위적으로 약간의 차이는 있으나 CopyMapTest, CopyTransferToTest, CopyTransferFromTest 가 나머지 3개보다 빠르것은 확인됨.

4. 소켓 채널

□ 소켓 채널

- 비블록킹 모드가 지원 가능
(기존 소켓이나 IO 를 이용한 N/W 프로그래밍의 최대 단점 : 블로킹)
 - : 클라이언트 커넥션 요구 받아들이는 서버소켓의 `accept()` 메소드 호출시
 - : 클라이언트와 연결된 소켓에서 데이터를 읽어들이기
 - 비블록킹 효과를 내기 위해서 멀티스레드 모델 사용
 - 스레드 개수가 많아지면 급격한 성능저하
- `SelectableChannel`을 상속해서 `Selector`와 함께 멀티플렉스(multiplex)IO가 가능

4. 소켓 채널

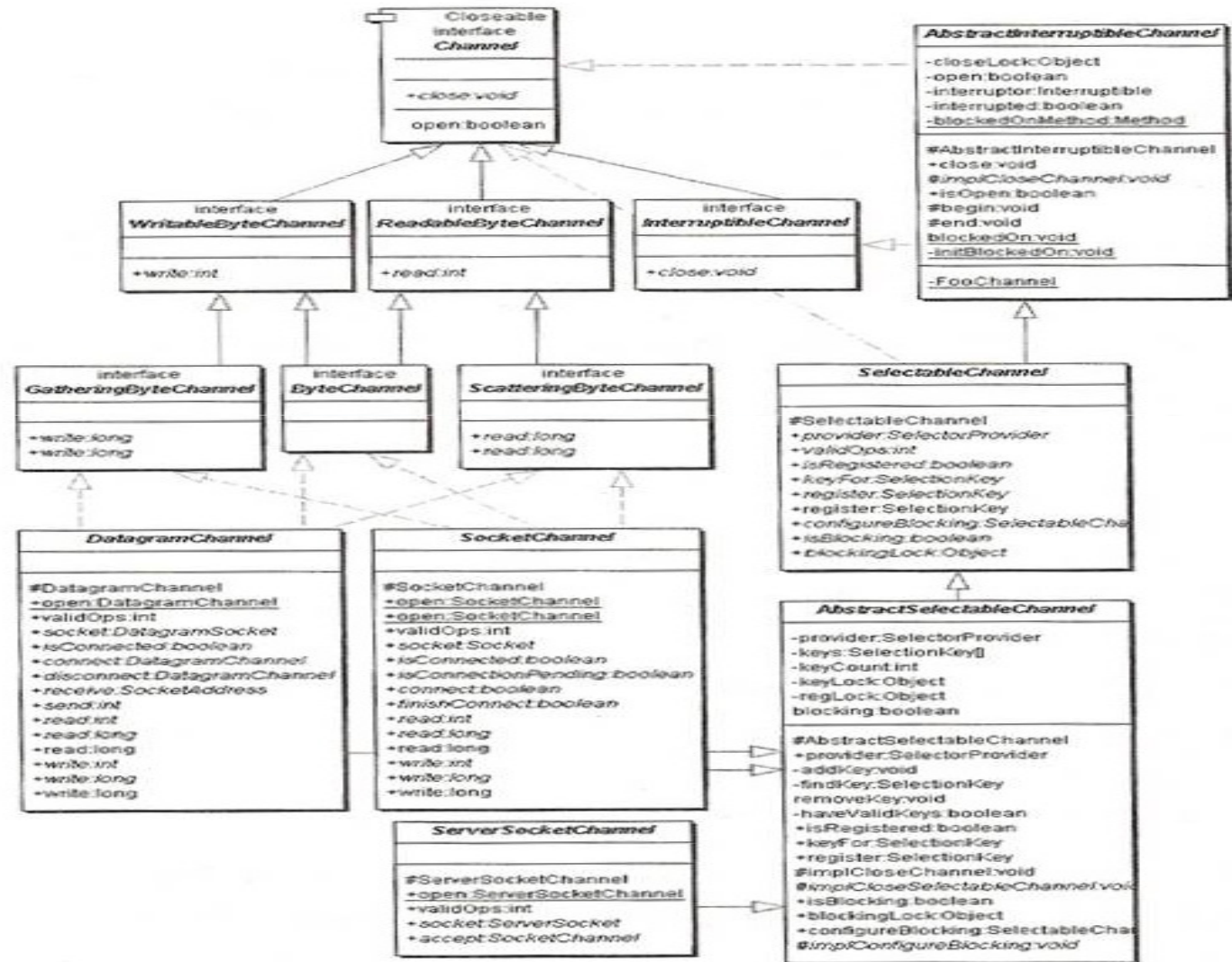
□ 소켓 채널

- 소켓 채널 클래스

- : ByteChannel과 ScatteringByteChannel, GatheringByteChannel 인터페이스 구현
- : 소켓 채널은 읽기, 쓰기가 모두 가능 / 효율적인 읽기, 쓰기를 위한 Scatter / Gather 도 이용
- : 서버 소켓채널은 스스로 데이터 전달 불가
(서버에 접속하려고 시도하는 클라이언트들을 처리하는데에만 사용)

4. 소켓 채널(계속)

□ 소켓 채널의 클래스



4. 소켓 채널(계속)

□ 비블록킹 모드

- 소켓채널 비블록킹 모드 설정 방법
 - : 수퍼클래스인 `SelectableChannel`에서 소켓채널을 비블록킹 모드로 설정하는 메소드 이용 (간단)
- `SelectableChannel` API

```
public abstract class SelectableChannel extends AbstractChannel implements
    Channel
{
    public abstract void configureBlocking (boolean block) throws
        IOException;
    public abstract boolean isBlocking();
    public abstract Object blocking();
}
```

4. 소켓 채널(계속)

□ 비블록킹 모드

- SelectableChannel API

: SelectableChannel 두가지 기능

1. 셀렉터(15장)와 함께 멀티플렉스 IO 작동을 하기 위해 소켓 채널을 셀렉터에 등록하는 것과 관련된 기능
2. 소켓 채널의 블로킹 모드 설정에 관련된 기능
3. 모든 소켓채널들은 기본적으로 블로킹 또는 비블로킹 모드 중 한가지 상태로 존재

: configureBlocking()

1. 파라미터 값으로 true 를 설정하면 블로킹 모드, false로 설정하면 비블로킹 모드로 설정

: isBlocking()

1. 소켓채널의 모드가 블로킹인지, 비블로킹인지 질의하는 메소드
2. 리턴값이 true면 블로킹모드, false 면 비블로킹 모드

4. 소켓 채널(계속)

□ 비블록킹 모드

- SelectableChannel API

: blockingLock()

1. 소켓채널의 블로킹 모드를 변경하는 것을 막을때 사용
2. 메소드는 Object 객체 리턴 / 리턴된 객체는 소켓 채널의 블로킹 모드를 변경하려고 할때 사용
3. 이 객체에 대해 오직 하나의 스레드가 락을 얻고 채널의 블로킹 모드를 변경할수 있도록 하는 방법을 사용
4. 락은 blockingLock() 통해 리턴된 객체를 synchronized 키워드의 대상으로 사용해서 얻는다.
5. 코드 임계영역이 실행되는 동안 다른 스레드에 의해 소켓 채널의 블로킹 모드가 변경되는 것을 막거나 다른 스레드에 영향을 주지 않고 소켓 채널의 블로킹 모드를 잠시 변경 할수 있게 한다.
6. 비블록킹 모드는 일반적으로 서버쪽에서 사용

4. 소켓 채널(계속)

□ 비블록킹 모드

- 소켓채널의 블록킹 모드를 변경한후 다시 원상태로 소켓채널의 블록킹 모드를 돌려놓는 과정

```
Object lockObj = serverSocketChannel.blockingLock();

// 임계영역안에서 처리하므로 다른 스레드에 영향을 주지 않지만
// 성능 저하를 가져오게 될 것이다.

synchronized (lockObj) {
    // 서버 소켓채널의 현재 모드를 저장한다.
    boolean preMode = serverSocketChannel.isBlocking();

    // 서버소켓 채널을 비블록킹 모드로 설정한다.
    serverSocketChannel.configureBlocking(false);

    // 접속한 클라이언트를 처리한다.
    registerClient (socket);

    // 서버소켓 채널을 저음 모드로 돌려놓는다.
    serverSocketChannel.configureBlocking(preMode);
}
```

4. 소켓 채널(계속)

□ 서버 소켓 채널

- 서버 소켓 채널의 API

```
public abstract class ServerSocketChannel extends AbstractSelectableChannel
{
    public static ServerSocketChannel open() throws IOException;
    public abstract ServerSocket socket();
    public abstract ServerSocket accept() throws IOException;
    public final int validOps();
}
```

: 기존의 서버소켓과 동일한 동작 (단 서버소켓 채널은 비블록킹 모드 동작)

: 서버 소켓 객체는 정적 메소드인 open() 통해 만듦 (new 사용 불가)

4. 소켓 채널(계속)

□ 서버 소켓 채널

- 서버 소켓 채널의 API

: 서버 소켓 채널은 `bind()` 미제공

: `open()` 통해 생성된 서버소켓 채널을 우리가 지정한 IP 와 바운딩 하는 방법

1. `open()` 호출 : 바운드 안된 서버 소켓, 연결된 소켓채널이 리턴
2. `socket()` 은 이렇게 연결된 서버 소켓을 리턴
3. `socket()` 을 통해 얻어진 서버소켓 객체의 `bind()` 메소드를 통해 메소드의 파라미터로 주어진 IP 와 포트에 바운드

4. 소켓 채널(계속)

□ 서버 소켓 채널

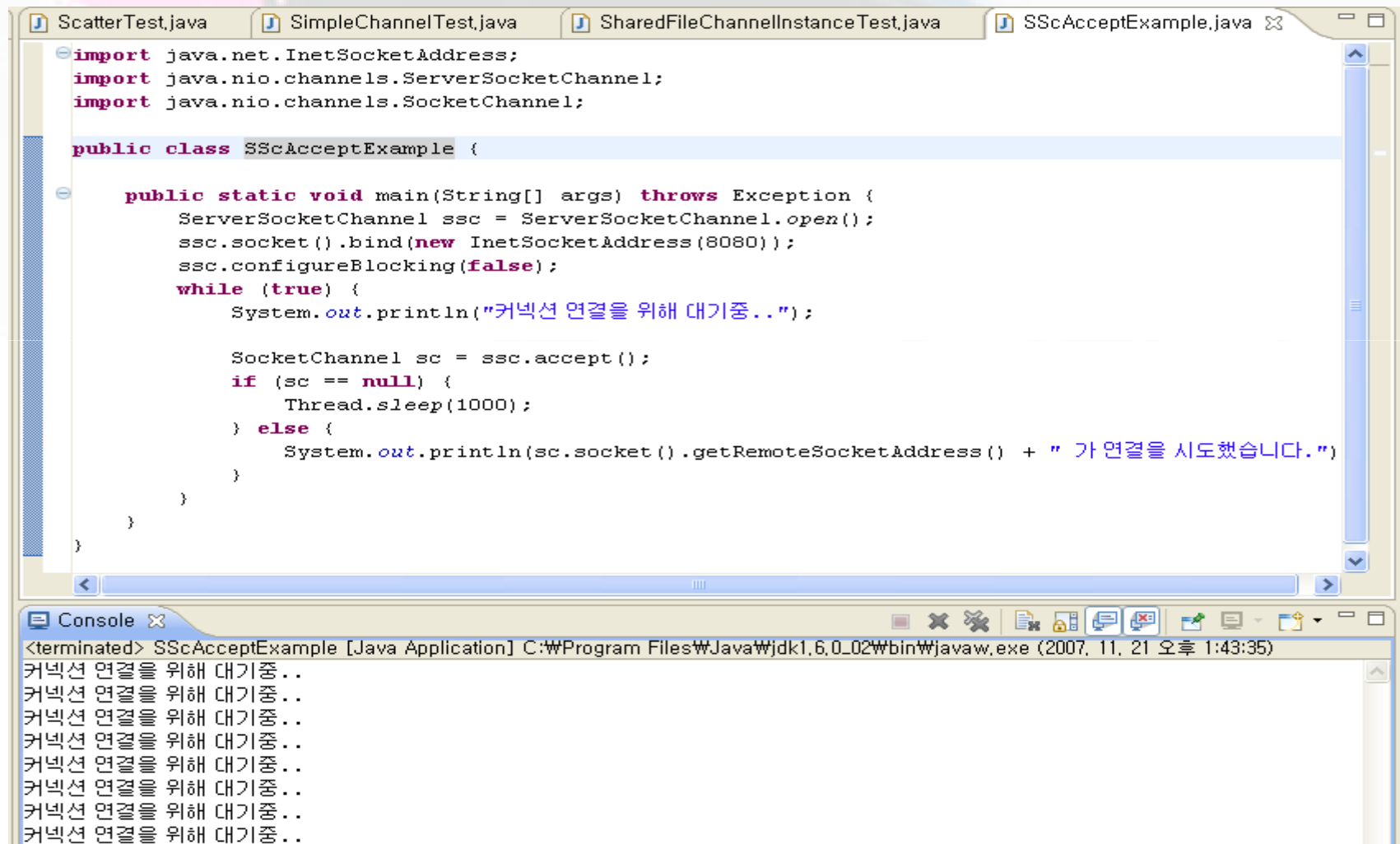
- 자신의 로컬 호스트 IP로 서버 소켓채널을 바인드 하는 코드 템플릿

```
ServerSocketChannel serverSocketChannel = ServerSocketChannel.open()  
ServerSocketChannel.configureBlocking(false);  
ServerSocket serverSocket = serverSocketChannel.socket();  
  
InetAddress ia = InetAddress.getLocalHost();  
// InetAddress ia = InetAddress.getByName( "192.168.0.2" );  
int port = 8080 ;  
InetAddress isa = new InetSocketAddress(ia, port);  
serverSocket.bind(isa);
```

- : 서버소켓을 통해 바인드한 후에 클라이언트의 접속을 처리(accept() 사용)
- : 서버소켓채널과 서버소켓 두가지 모두 accept() 통해 클라이언트 접속처리
- : 서버소켓 (블록킹 모드 처리), 서버채널 소켓(비블록킹 / 블록킹 모드처리)

4. 소켓 채널(계속)

□ 서버 소켓 채널(예제)



The screenshot shows an IDE with four tabs: ScatterTest.java, SimpleChannelTest.java, SharedFileChannelInstanceTest.java, and SScAcceptExample.java. The SScAcceptExample.java tab is active, displaying the following Java code:

```
import java.net.InetSocketAddress;
import java.nio.channels.ServerSocketChannel;
import java.nio.channels.SocketChannel;

public class SScAcceptExample {

    public static void main(String[] args) throws Exception {
        ServerSocketChannel ssc = ServerSocketChannel.open();
        ssc.socket().bind(new InetSocketAddress(8080));
        ssc.configureBlocking(false);
        while (true) {
            System.out.println("커넥션 연결을 위해 대기중..");

            SocketChannel sc = ssc.accept();
            if (sc == null) {
                Thread.sleep(1000);
            } else {
                System.out.println(sc.socket().getRemoteSocketAddress() + " 가 연결을 시도했습니다.")
            }
        }
    }
}
```

Below the code editor is a console window titled "Console" showing the output of the program:

```
<terminated> SScAcceptExample [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 11. 21 오후 1:43:35)
커넥션 연결을 위해 대기중..
커넥션 연결을 위해 대기중..
커넥션 연결을 위해 대기중..
커넥션 연결을 위해 대기중..
커넥션 연결을 위해 대기중..
커넥션 연결을 위해 대기중..
커넥션 연결을 위해 대기중..
커넥션 연결을 위해 대기중..
커넥션 연결을 위해 대기중..
```


4. 소켓 채널(계속)

□ 소켓 채널

```
public abstract class SocketChannel extends AbstractSelectableChannel
    implements ByteChannel, ScatteringByteChannel, GatheringByteChannel {

    // 소켓채널 생성 및 소켓 채널과 연관된 소켓을 구하기 위한 메소드들
    public static SocketChannel open() throws IOException
    public static SocketChannel open(SocketAddress remote) throws IOException
    public abstract Socket socket()

    // 소켓 채널의 연결에 관련된 메소드들
    public abstract boolean connect(SocketAddress remote) throws IOException
    public abstract boolean isConnectionPending()
    public abstract boolean finishConnect() throws IOException
    public abstract boolean isConnected()

    // 선택터와 연관된 것으로 15장에서 자세히 살펴볼 것이다.
    public final int validOps()
```

- read (), write () 메소드 제외 (Scatter, Gather와 파일 채널에서의 사용방법과 동작 방식 유사)

4. 소켓 채널(계속)

□ 소켓 채널

- 소켓 채널 API

: AbstractSelectableChannel 상속

(SelectableChannel 이 제공해주는 비블록킹 모드 사용 가능)

: ByteChannel, ScatteringByteChannel, GatheringByteChannel 구현

(읽기 쓰기 모두 지원하는 양방향성 특징 / 운영체제의 네이티브 코드의 도움을 받는 효율적인 읽기, 쓰기 가능)

: 정적 메소드인 open() 메소드를 통해 생성

(소켓채널은 연관된 소켓을 함께 생성)

4. 소켓 채널(계속)

□ 소켓 채널

- 소켓 채널 생성시 주의할점

- : 소켓채널을 생성할 경우에는 항상 소켓채널과 연결된 소켓이 같이 생성
- : 반대인 소켓을 생성할 경우에는 소켓 채널 생성되지 않는다.

```
Socket so = new Socket() ;  
SocketChannel sc = so.getChannel();  
  
if (sc == null) {  
    System.out.println( "Null" );  
}else {  
    System.out.println( "Not NULL" );  
}
```

: 출력값 - Null

4. 소켓 채널(계속)

□ 소켓 채널

```
SocketChannel sc =  
SocketChannel.open(new InetSocketAddress( "host IP" , port));
```

- 소켓채널은 또한 서버채널과 달리 연결할 대상을 지정할수 있는 open() 메소드를 제공
- 소켓채널을 생성함과 동시에 원격지의 서버로 연결
- 위의 템플릿은 다음과 같은 동작을 한다.

```
SocketChannel sc = SocketChannel.open();  
sc.connect(new InetSocketAddress( "host Ip" , port));
```

- 소켓채널은 소켓과 마찬가지로 원격지의 서버와 연결하기 위한 connect() 메소드를 제공
- 위의 코드는 모두 블로킹 모드의 소켓채널이 생성되어 서버에 연결

4. 소켓 채널(계속)

□ 소켓 채널

- 비블록킹 모드로 서버와 연결하는 방법

```
SocketChannel sc =  
SocketChannel.open(new InetSocketAddress( "host IP" , port));  
sc.configureBlocking(false);
```

```
SocketChannel sc = SocketChannel.open()  
sc.configureBlocking(false);  
sc.connect(new InetSocketAddress ( "host IP" ,port)) ;  
sc.finishConnect();
```

: 반드시 finishConnect() 메소드 호출, 만약 호출하지 않으며 소켓채널의 연결시도는 대기상태로 있게 된다.

: finishConnect() – 비블록킹 모드에서 연결을 마무리하는 역할

4. 소켓 채널(계속)

□ 소켓 채널

- isConnetcionPending()

: 채널이 현재 서버에 연결을 진행하고 있는 중인지를 질의하는 메소드

```
SocketChannel sc = SocketChannel.open()
sc.configureBlocking(false);

System.out.println( "Is ConnectionPending 1 : " + sc.isConnectionPending());

// 연결을 시도한다.
sc.connect(new InetSocketAddress ( "host IP" ,port)) ;
System.out.println( "Is ConnectionPending 2 : " +sc.isConnectionPending());
// 연결을 마무리한다.

sc.finishConnect();
System.out.println( "Is ConnectionPending 3 : " +sc.isConnectionPending());
```

4. 소켓 채널(계속)

□ 소켓 채널

- isConnected()

: 결과가 true 면 연결이 이미 이루어진 상태

: 결과가 false 면 연결이 안된 상태

- validOps()

: 셀렉터와 관련된 메소드 (15장 에서 구체적으로 다룸)

4. 소켓 채널(계속)

□ 소켓 채널

- 소켓채널이 서버에 연결하는 예제

```
import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.nio.channels.SocketChannel;

public class SCConnectionTest {

    private static int PORT = 80;

    public static void main(String[] args) throws Exception {
        InetAddress ia = InetAddress.getLocalHost();
        InetSocketAddress isa = new InetSocketAddress(ia, PORT);

        SocketChannel sc = SocketChannel.open();
        sc.configureBlocking(false);
        System.out.println("Is ConnectionPending 1 : " + sc.isConnectionPending());
        sc.connect(isa);
        System.out.println("Is ConnectionPending 2 : " + sc.isConnectionPending());
        sc.finishConnect();
        System.out.println("Is ConnectionPending 3 : " + sc.isConnectionPending());

        // 또는 다음을 사용해도 같은 결과를 얻을 수 있음..
        SocketChannel sc2 = SocketChannel.open(isa);
        sc2.configureBlocking(false);

        System.out.println("Is Connected : " + sc.isConnected());
        System.out.println("Is Blocking Mode : " + sc.isBlocking());
    }
}
```

Console

<terminated> SCConnectionTest [Java Application] C:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe (2007. 11. 06 오후 8:01:28)

```
Is ConnectionPending 1 : false
Is ConnectionPending 2 : true
Is ConnectionPending 3 : true
Is Connected : false
Is Blocking Mode : false
```


4. 소켓 채널(계속)

□ 데이터그램 채널(데이터 그램 소켓 채널)

- 데이터그램 소켓(DatagramSocket)과 연관관계
- 소켓 채널은 TCP / IP 같은 커넥션 기반의 스트림 프로토콜 모델
- 데이터그램 채널은 UDP / IP 같은 패킷 기반의 프로토콜 모델
- 데이터그램 채널의 API

```
public abstract class DatagramChannel extends AbstractChannel implements
    ByteChannel, GatheringByteChannel, ScatteringByteChannel {

    public static DatagramChannel open() throws IOException
    public abstract DatagramSocket socket()

    public abstract DatagramChannel connect(SocketAddress remote)
    throws IOException
    public abstract boolean isConnected()
    public abstract DatagramChannel disconnect() throws IOException

    public abstract SocketAddress receive (ByteBuffer dst) throws IOException
    public abstract int send(ByteBuffer src, SocketAddress target)
    throws IOException
}
```

4. 소켓 채널(계속)

□ 데이터그램 채널(데이터 그램 소켓 채널)

- 데이터그램 채널의 API

: read(), write() 메소드는 제외

(사용하는 방법은 Scatter/Gather 과 파일채널에서의 사용방법 유사)

: AbstractSelectableChannel 상속

(SelectableChannel 이 제공해주는 비블록킹 모드 사용 가능)

: ByteChannel, ScatteringByteChannel, GatheringByteChannel 구현

(읽기 쓰기 모두 지원하는 양방향성 특징 / 운영체제의 네이티브 코드의 도움을 받는 효율적인 읽기, 쓰기 가능)

ACHROMATIC COLOR

4. 소켓 채널(계속)

□ 데이터그램 채널

- 서버소켓채널과 같이 채널 생성을 위해 파라미터 없는 open() 메소드 한 개만이 존재
- 다음 방식과 같이 데이터그램채널을 바인드

```
DatagramChannel datagramChannel = DatagramChannel.open();  
DatagramSocket datagramSocket = datagramChannel.socket();  
datagramSocket.bind(new InetSocketAddress( "host IP" , port));
```

: open() 생성하는 의미

1. 데이터그램채널과 한쌍이 되는 데이터그램소켓이 함께 생성
2. 데이터그램채널이 연결되지 않은 상태여서 바인드 해야 함

4. 소켓 채널(계속)

□ 데이터그램 채널

```
public abstract SocketAddress receive (ByteBuffer dst) throws IOException  
public abstract int send(ByteBuffer src, SocketAddress target) throws IOException
```

- 패킷에는 패킷을 받아볼 목적지의 주소와 데이터 포함(비연결지향)
- receive() 메소드
 - : 패킷을 읽어들이는 메소드
 - : 데이터그램 채널이 블로킹 모드
 - => 읽어 들일 패킷이 없으면 블로킹되어 있다가 읽어 들일 데이터가 있다면 즉시 패킷 내용을 파라미터로 주어진 ByteBuffer로 읽어 들이고 해당 패킷을 보낸 쪽의 주소에 리턴
 - : 데이터 그램 채널이 비블로킹 모드
 - => 읽어들이 패킷이 없으면 즉시 null을 리턴 하지만 읽어들이 데이터가 있을경우 블로킹 모드와 동일하게 동작
- send() 메소드
 - : 데이터그램 채널이 원격지의 디바이스로 패킷을 보내는데 사용 (src 내용과 두번째 파라미터 목적지 주소를 합쳐서 N/W 보냄)

4. 소켓 채널(계속)

□ 데이터그램 채널

```
public abstract boolean isConnected()  
public abstract DatagramChannel disconnect() throw IOException
```

- connect 메소드를 사용해서 데이터그램채널을 연결했는지의 여부를 알아보기 위한 isConnected() 메소드와 연결을 끊기위한 disconnect() 메소드가 있다.
- 원격지의 주소에 연결되지 않는 상태에서 read(), write() 사용 못함 (전송할 목적지의 주소와 함께 데이터를 패킷으로 묶어서 보내기 때문)
- 접속이 이루어지지 않은 상황에서는 receive(), send() 메소드 사용
- connect() 메소드로 데이터그램채널이 원격지의 어떤 주소에 연결된 상태이어야 한다.

4. 소켓

□ 예제 14. UDPEchoServer

```
public class UDPEchoServer
{
    protected int port;

    public UDPEchoServer(int port)
    {
        this.port = port;
    }

    public void execute() throws IOException, InterruptedException
    {
        //비블록킹 모드로 데이터그램채널을 로컬 호스트 8080포트에 바인드시킨다.
        DatagramChannel channel = DatagramChannel.open();
        channel.socket().bind(new InetSocketAddress("localhost", port));
        channel.configureBlocking(false);

        ByteBuffer buffer = ByteBuffer.allocateDirect(1024);
        while(true)
        {
            buffer.clear();
            //패킷을 받는다.
            SocketAddress addr = channel.receive(buffer);
            //만약, 도착된 패킷이 있다면 패킷에 포함된 메시지를
            //다시 보낸 쪽으로 돌려보낸다.
            if(addr != null)
            {
                System.out.println("패킷이 도착했습니다.");
                buffer.flip();
                channel.send(buffer, addr);
                //만약 패킷이 도착하지 않았다면 5초간 멈춘다.
            }else{
                System.out.println("도착한 패킷이 없습니다.");
                Thread.sleep(5000);
            }
        }
    }

    public static void main(String[] args) throws IOException, InterruptedException
    {
        UDPEchoServer server = new UDPEchoServer(8080);
        server.execute();
    }
}
```

4. 소켓 채널(계속)

□예제 14-18 UDPEchoClient

```
public class UDPEchoClient
{
    private static Timer timer = null;
    public UDPEchoClient(int seconds) throws Exception
    {
        timer = new Timer();
        //2초마다 EchoClientTask를 수행한다.
        timer.schedule(new EchoClientTask(), seconds);

        //10초간 멈춘다.
        Thread.sleep(10000);
        //EchoClientTask가 멈춘다.
        timer.cancel();
    }
}
```

```
class EchoClientTask extends TimerTask
{
    public void run()
    {
        try{
            //블록킹 모드의 데이터그램채널을 생성한다.
            DatagramChannel channel = DatagramChannel.open();
            //Echo 서버의 주소 생성 및 다이렉트 ByteBuffer를 생성한다.
            SocketAddress sa = new InetSocketAddress("localhost", 8080);
            ByteBuffer buffer = ByteBuffer.allocateDirect(1024);

            while(!Thread.interrupted())
            {
                //버퍼를 clear시킨 후에 버퍼에 데이터를 집어넣는다.
                buffer.clear();
                buffer.put("데이터그램채널 테스트..".getBytes());
                buffer.flip();
                //Echo 서버에 해당 버퍼의 메시지를 보낸다.
                channel.send(buffer, sa);

                //버퍼를 재사용하기 위해 clear한다.
                buffer.clear();
                //서버로부터 메시지를 받는다.
                SocketAddress addr = channel.receive(buffer);
                buffer.flip();

                //받은 메시지를 출력한다.
                byte[] bb = new byte[buffer.limit()];
                buffer.get(bb, 0, buffer.limit());
                String data = new String(bb);
                System.out.println("Receive: " + data);

                //1초간 멈춘다.
                Thread.sleep(1000);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

public static void main(String[] args) throws Exception
{
    new UDPEchoClient(2);
}
```