19장 NIO 기반 입출력 및 네트워킹

19.1 NIO 소개

- 기존 java.io API와 다른 새로운 입출력 API
- 자바4부터 추가 → 자바7부터 네트워크 지원 강화된 NIO.2 API 추가
- 관련 패키지

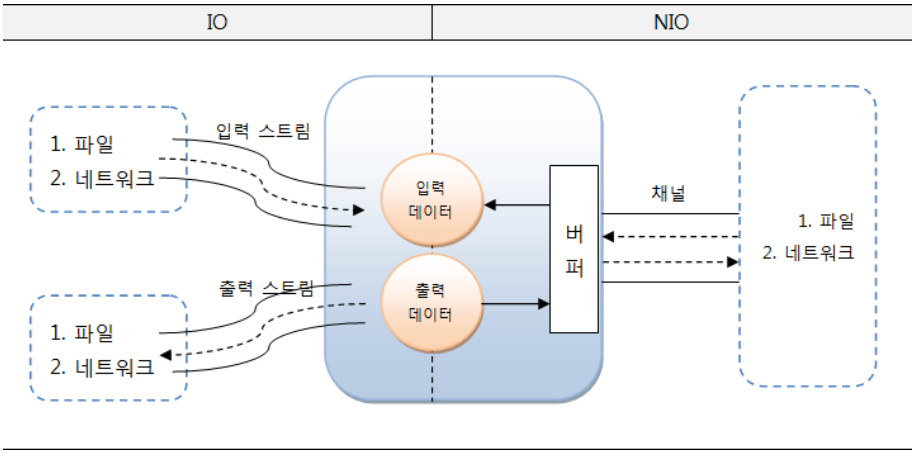
NIO 패키지	포함되어 있는 내용
java.nio	다양한 버퍼 클래스
java.nio.channels	파일 채널, TCP 채널, UDP 채널등의 클래스
java.nio.channels.spi	java.nio.channels 패키지를 위한 서비스 제공자 클래스
java.nio.charset	문자셋, 인코더, 디코더 API
java.nio.charset.spi	java.nio.charset 패키지를 위한 서비스 제공자 클래스
java.nio.file	파일 및 파일 시스템에 접근하기 위한 클래스
java.nio.file.attribute	파일 및 파일 시스템의 속성에 접근하기 위한 클래스
java.nio.file.spi	java.nio.file 패키지를 위한 서비스 제공자 클래스

19.1.1 IO와 NIO의 차이점

구분	IO	NIO
입출력 방식	스트림 방식	채널 방식
버퍼 방식	넌버퍼(non-buffer)	버퍼(buffer)
동기 / 비동기 방식	동기 방식	동기 / 비동기 방식 모두 지원
블로킹 / 넌블로킹 방식	블로킹 방식	블로킹 / 넌블로킹 방식 모두 지원

(1) 스트림 vs. 채널

- IO 스트림 : 입력 스트림과 출력 스트림으로 구분되어 별도 생성
- NIO 채널 : 양방향으로 입출력이 가능하므로 하나만 생성



(2) 넌버퍼 vs. 버퍼

- IO 스트림 : 넌버퍼(non-buffer)
  - IO에서는 1바이트씩 읽고 출력 -> 느림
  - 보조 스트림인 BufferedInputStream, BufferedOutputStream를 사용해 버퍼 제공 가능
  - 스트림으로부터 입력된 전체 데이터를 별도로 저장해야한다. 저장 후 입력 데이터의 위치 이동해가면서 자유롭게 이용 가능

- NIO 채널 : 버퍼(buffer)
  - 기본적으로 버퍼 사용해 입출력 -> 성능 좋음
  - 읽은 데이터를 무조건 버퍼(Butter: 메모리 저장소)에 저장 -> 버퍼 내에서 데이터 위치 이동해 가며 필요한 부분만 읽고 쓸 수 있음

(3) 블로킹 vs. 넌블로킹

- IO 스트림 : 블로킹
  - 입력 스트림의 read() 메소드 호출 -> 데이터 입력 전까지 스레드는 블로킹(대기상태)
  - 출력 스트림의 write() 메소드 호출 -> 데이터 출력 전까지 스레드는 블로킹
  - 스레드 블로킹 -> 다른 일을 할 수가 없고 interrupt 해 블로킹 빠져나올 수도 없음
  - 블로킹을 빠져 나오는 유일한 방법 -> 스트림을 닫는 것

- NIO 채널 : 블로킹, 넌블로킹
  - NIO 블로킹은 스레드를 interrupt 함으로써 빠져나올 수 있음
  - NIO는 넌블로킹 지원 -> 입출력 작업 시 스레드가 블로킹되지 않음
  - NIO 넌블로킹의 핵심 객체는 멀티플렉서인 셀렉터(Selector)이다. 셀렉터는 복수 개의 채널 중에서 준비 완료된 채널을 선택하는 방법을 제공해준다.

19.1.2 IO와 NIO의 선택

- NIO는 연결 클라이언트 수가 많고, 하나의 입출력 처리 작업이 오래 걸리지 않는 경우에 사용하는 것이 좋다.

- IO는 대용량 데이터를 처리할 경우에 유리하다.

19.2 파일과 디렉토리

19.2.1 경로 정의(Path)

- java.nio.file.Path 인터페이스
  - IO의 java.io.File 클래스에 대응
  - NIO의 여러 곳에서 파일 경로 지정 위해 Path 사용

- Path 구현 객체
  - java.nio.file.Paths 클래스의 정적 메소드인 get() 메소드로 얻음

```
Path path = Paths.get("C:/Temp/file.txt");
Path path = Paths.get("C:/Temp", "file.txt");
```

19.2.2 파일 시스템 정보(FileSystem)

- 운영체제의 파일 시스템은 FileSystem인터페이스 통해 접근
- FileSystem 구현 객체는 FileSystems의 정적 메소드인 getDefault()로 얻을 수 있음
- FileSystem에서 제공하는 메소드

리턴타입	메소드(매개변수)	설명
Iterable<FileStore>	getFileStores()	드라이버 정보를 가진 FileStore 객체들을 리턴
Iterable<Path>	getRootDirectories()	루트 디렉토리 정보를 가진 Path 객체들을 리턴
String	getSeparator()	디렉토리 구분자 리턴

```
FileSystem fileSystem = FileSystems.getDefault();
```

[FileSystemExample.java] 파일 시스템 정보 얻기

```
package sec02.exam02_filesystem;

import java.nio.file.FileStore;
import java.nio.file.FileSystem;
import java.nio.file.FileSystems;
import java.nio.file.Path;

public class FileSystemExample {
	public static void main(String[] args) throws Exception {
		FileSystem fileSystem = FileSystems.getDefault();
		for(FileStore store : fileSystem.getFileStores()) {
			System.out.println("드라이버명: " + store.name());
			System.out.println("파일시스템: " + store.type());
			System.out.println("전체 공간: \t\t" + store.getTotalSpace() + " 바이트");
			System.out.println("사용 중인 공간: \t" + (store.getTotalSpace() -
store.getUnallocatedSpace()) + " 바이트");
			System.out.println("사용 가능한 공간: \t" + store.getUsableSpace() + "
바이트");
			System.out.println();
		}

		System.out.println("파일 구분자: " + fileSystem.getSeparator());
		System.out.println();

		for(Path path : fileSystem.getRootDirectories()) {
			System.out.println(path.toString());
		}
	}
}
```

19.2.3 파일 속성 읽기 및 파일, 디렉토리 생성/삭제

- java.nio.file.Files 클래스
  - 파일과 디렉토리의 생성 및 삭제, 이들의 속성을 읽는 메소드 제공

[FileExample.java] 파일 속성 얻기

```
package sec02.exam03_file_directory;

import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class FileExample {
	public static void main(String[] args) throws Exception {
		Path path = Paths.get("src/sec02/exam03_file_directory/FileExample.java");
		System.out.println("디렉토리 여부: " + Files.isDirectory(path));
		System.out.println("파일 여부: " + Files.isRegularFile(path));
		System.out.println("마지막 수정 시간: " + Files.getLastModifiedTime(path));
		System.out.println("파일 크기: " + Files.size(path));
		System.out.println("소유자: " + Files.getOwner(path).getName());
		System.out.println("숨김 파일 여부: " + Files.isHidden(path));
		System.out.println("읽기 가능 여부: " + Files.isReadable(path));
		System.out.println("쓰기 가능 여부: " + Files.isWritable(path));
	}
}
```

[DirectoryExample.java] 디렉토리 내용 얻기

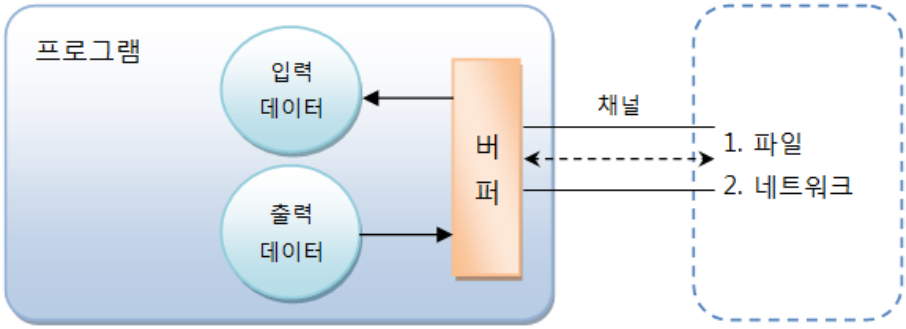
```
package sec02.exam03_file_directory;

import java.nio.file.DirectoryStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class DirectoryExample {
	public static void main(String[] args) throws Exception {
		Path path1 = Paths.get("C:/Temp/dir/subdir");
		Path path2 = Paths.get("C:/Temp/file.txt");

		if(Files.notExists(path1)) {
			Files.createDirectories(path1); // 경로상에 존재하지 않는 모든 디렉토리 생성
		}

		if(Files.notExists(path2)) {
			Files.createFile(path2); // 파일 생성
		}

		Path path3 = Paths.get("C:/Temp");
		// 디렉토리 내용을 스트림(반복자)으로 얻음
		DirectoryStream<Path> directoryStream = Files.newDirectoryStream(path3);
		for(Path path : directoryStream) {
			if(Files.isDirectory(path)) {
				System.out.println("[디렉토리] " + path.getFileName());
			} else {
				System.out.println("[파일] " + path.getFileName() + " (크기:" +
Files.size(path) + ")");
			}
		}
	}
}
```

19.2.4 와치 서비스(WatchService)

- 디렉토리의 내용 변화 감시
- 자바 7에서 처음 소개
- 디렉토리내의 파일의 생성, 삭제, 수정 감시
- 적용 사례
  - 에디터 바깥에서 파일 내용 수정하면 파일 내용이 변경됐으니 파일을 다시 불러올 것인지 묻는 대화상자 띄움

19.3 버퍼

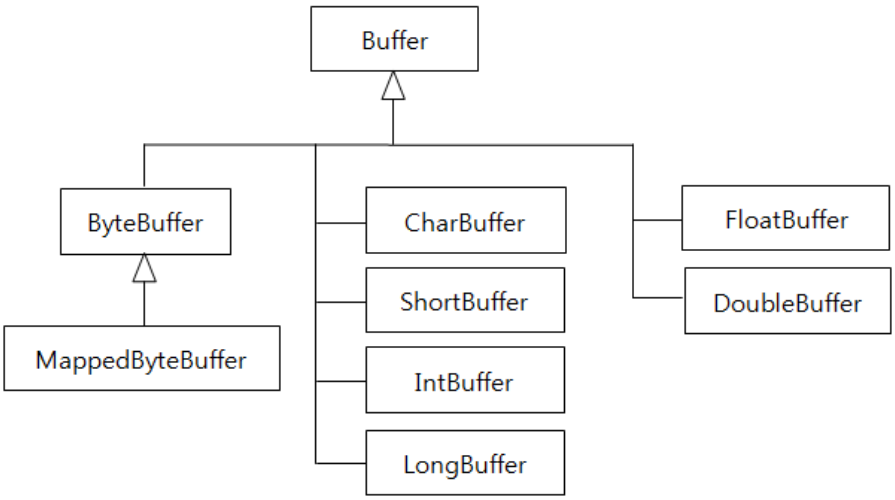
- 버퍼는 읽고 쓰기 가능한 메모리 배열
- NIO에서는 데이터를 입출력을 하기 위해서는 항상 버퍼 사용



19.3.1 Buffer 종류

(1) 데이터 타입에 따른 버퍼

- NIO 버퍼는 저장되는 데이터 타입에 따라 별도의 클래스로 제공된다.
- 이 버퍼 클래스들은 Buffer 추상 클래스를 모두 상속하고 있다.
- MappedByteBuffer는 파일의 내용에 랜덤하게 접근하기 위해서 파일의 내용을 메모리와 맵핑시킨 버퍼이다.



(2) 넌다이렉트와 다이렉트 버퍼

■ 어떤 메모리를 사용하느냐에 따른 분류

구분	넌다이렉트 버퍼	다이렉트 버퍼
사용하는 메모리 공간	JVM 의 힙 메모리	운영체제의 메모리
버퍼 생성 시간	버퍼 생성이 빠르다.	버퍼 생성이 느리다.
버퍼의 크기	작다.	크다.(큰 데이터를 처리할 때 유리)
입출력 성능	낮다.	높다.(입출력이 빈번할 경우 유리)

[BufferSizeExample.java] 넌다이렉트와 다이렉트 크기 비교

```
package sec03.exam01_direct_buffer;

import java.nio.ByteBuffer;

public class BufferSizeExample {
	public static void main(String[] args) {
		ByteBuffer directBuffer = ByteBuffer.allocateDirect(200 * 1024 * 1024);
		System.out.println("다이렉트 버퍼가 생성되었습니다.");

		ByteBuffer nonDirectBuffer = ByteBuffer.allocate(200 * 1024 * 1024);
		System.out.println("넌다이렉트 버퍼가 생성되었습니다.");
	}
}
```

19.3.2 Buffer 생성

■ 각 데이터 타입별로 넌다이렉트 버퍼를 생성하기 위해서는 각 Buffer 클래스의 allocate()와 wrap() 메소드를 호출하면 되고, 다이렉트 버퍼는 ByteBuffer의 allocateDirect() 메소드를 호출하면 된다.

(1) allocate() 메소드

■ 각 데이터 타입 별 넌다이렉트 버퍼 생성

■ 매개값 - 해당 데이터 타입의 저장 개수

```
ByteBuffer byteBuffer = ByteBuffer.allocate(100);
CharBuffer charBuffer = CharBuffer.allocate(100);
```

(2) wrap() 메소드

■ 이미 생성되어 있는 타입 별 배열을 래핑해 버퍼 생성

■ 일부 데이터만 가지고도 버퍼 생성 가능!

```
byte[] byteArray = new byte[100];
ByteBuffer byteBuffer = ByteBuffer.wrap(byteArray);
char[] charArray = new char[100];
CharBuffer charBuffer = CharBuffer.wrap(charArray);

ByteBuffer byteBuffer = ByteBuffer.wrap(byteArray, 0, 50); // 0인텍스부터 50개만 버퍼로 생성한다.
```

(3) allocateDirect() 메소드

■ JVM 힙 메모리 바깥쪽 (운영체제가 관리하는 메모리)에 다이렉트 버퍼 생성

■ 각 타입 별 Buffer 클래스에는 없고 ByteBuffer 에서만 제공

■ asXXXBuffer() 메소드(각 타입 별 다이렉트 버퍼 생성)
- asCharBuffer(), asShortBuffer(), asIntBuffer(), asLongBuffer(), asFloatBuffer(), asDoubleBuffer()
- 우선 다이렉트 ByteBuffer를 생성하고 호출
- 초기 다이렉트 ByteBuffer 생성 크기에 따라 저장 용량 결정

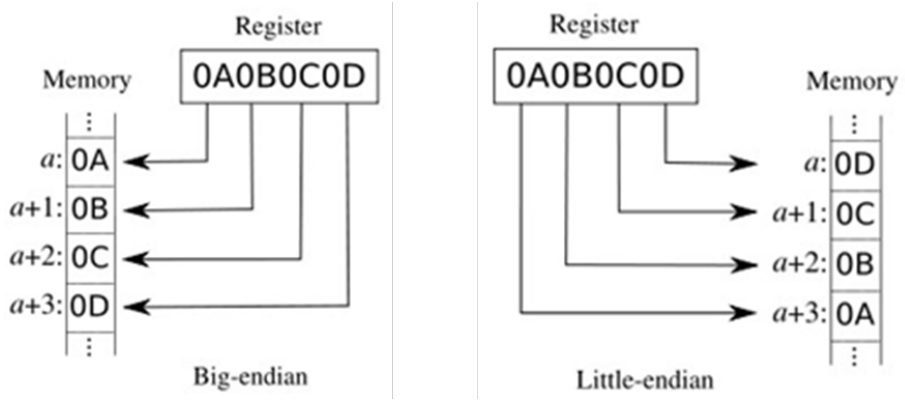
```
ByteBuffer byteBuffer = ByteBuffer.allocateDirect(100); //100개의 byte값 저장
CharBuffer charBuffer = ByteBuffer.allocateDirect(100).asCharBuffer(); //50개의 char값 저장
IntBuffer intBuffer = ByteBuffer.allocateDirect(100).asIntBuffer(); //25개의 int값 저장
```

(4) byte 해석 순서(ByteOrder)

■ 데이터를 처리할 때 바이트 처리 순서는 운영체제마다 차이가 있다.

■ 이러한 차이는 데이터를 다른 운영체제로 보내거나 받을 때 영향을 미치기 때문에 데이터를 다루는 버퍼도 이를 고려해야 한다.

■ Big endian(앞 바이트부터 먼저 처리) - Unix, JVM; Little endian(뒤 바이트부터 먼저 처리) - Windows, Linux



- JVM은 동일한 조건으로 클래스 실행해야 하므로 무조건 Big endian.
- Little endian으로 동작하는 운영체제에서 만든 데이터 파일을 Big endian로 동작하는 운영체제에서 읽어 들여야 한다면 ByteOrder 클래스로 데이터 순서를 맞춰야
- 운영체제와 JVM의 바이트 해석 순서가 다를 경우
  - JVM이 운영체제와 데이터 교환 할 때 자동 처리해주기 때문에 문제는 없다.
- 다이렉트 버퍼를 이용할 경우 운영체제의 native I/O를 사용
  - 운영체제의 기본 해석 순서로 JVM의 해석 순서를 맞추는 것이 성능에 도움

```
ByteBuffer byteBuffer = ByteBuffer.allocateDirect(100).order(ByteOrder.nativeOrder());
```

19.3.3 Buffer의 위치 속성

(1) Buffer의 네 가지 위치 속성

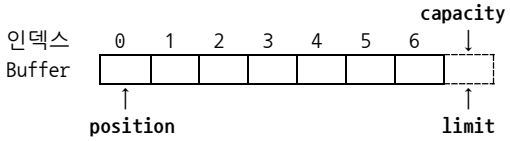
속성	설명
position	현재 읽거나 쓰는 위치값이다. 인덱스 값이기 때문에 0 부터 시작하며, 음수를 가지거나 limit 보다 큰값을 가질 수 없다. 만약 position 과 limit 의 값이 같아진다면 더이상 데이터를 쓰거나 읽을 수 없다는 뜻이 된다.
limit	버퍼에서 읽거나 쓸 수 있는 위치의 한계를 나타낸다. 이 값은 capacity 보다 작거나 같은 값을 가지며 음수 값은 가지지 않는다. 인덱스 값이기 때문에 0 부터 시작하며 최초 버퍼를 만들었을때는 capacity 와 같은 값을 가진다.
capacity	버퍼의 최대 데이터 개수(메모리 크기)를 나타낸다. 크기는 음수가 되지 않으며 한번 만들어지면 절대 변하지 않는다. 인덱스 값이 아니라 수량임을 주의하자.
mark	reset() 메소드를 실행했을 때에 돌아오는 위치를 지정하는 인덱스로서 mark() 메소드로 지정할 있다. 주의할 점은 반드시 position 이하의 값으로 지정해 주어야 한다. position 나 limit 의 값이 mark 값보다 작은 경우, mark 는 자동 제거된다. mark 없는 상태에서 reset() 메소드를 호출하면 InvalidMarkException 이 발생한다.

(2) 위치 속성의 크기 관계

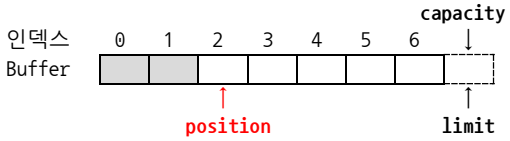
$$0 \le mark \le position \le limit \le capacity$$

(3) 위치 속성의 쓰기 모드 예

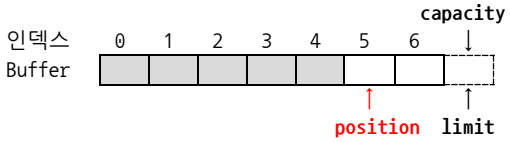
■ 7바이트 크기의 버퍼가 있다고 가정해보자.



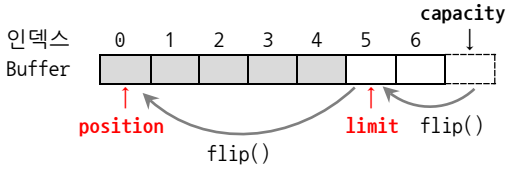
■ 먼저 2바이트를 버퍼에 저장해보자.



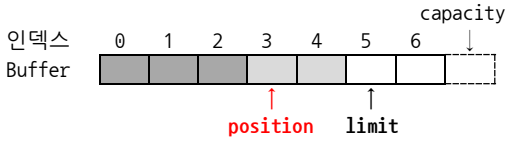
■ 계속해서 3바이트를 저장해보자.



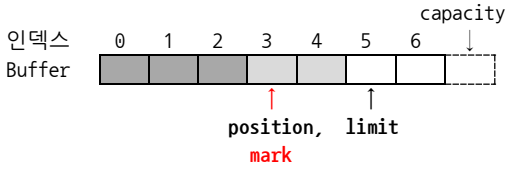
■ 이제 버퍼에 저장되어 있는 바이트를 읽어보자. 먼저 flip() 메소드를 호출해야 한다. flip() 메소드를 호출하면 limit을 현재 position 5 인덱스로 설정하고, position을 0번 인덱스로 설정한다.



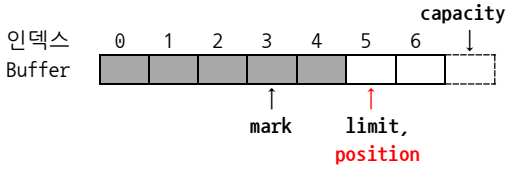
- 버퍼에서 3바이트를 읽는다고 가정해보자.



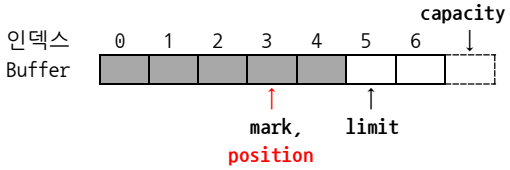
- position이 3번 인덱스를 가르키고 있을 때 mark() 메소드를 호출해서 현재 위치를 기억시켜 놓는다.



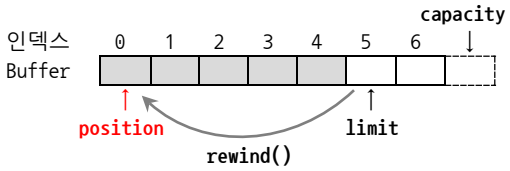
- 이어서 2바이트를 더 읽어보자.



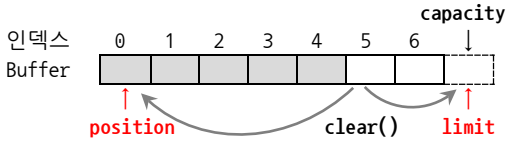
- 이번에는 position을 mark 위치로 다시 이동해야 한다고 가정해보자. reset() 메소드를 호출하며 position은 mark가 있는 3번 인덱스로 이동한다.



- 버퍼를 되감아 동일한 데이터를 한 번 더 읽고 싶다고 가정해보자. rewind() 메소드를 호출하면 limit은 변하지 않지만 position은 0번 인덱스로 다시 설정된다.



- 만약 rewind() 대신 clear() 메소드를 호출하면 Buffer의 세 가지 속성은 초기화된다. limit은 capacity로, position은 0으로 설정되고 mark는 자동적으로 없어진다. 하지만 데이터는 삭제되지 않는다.



19.3.4 Buffer 메소드

(1) 공통 메소드

- Buffer 추상 클래스에 정의된 메소드

리턴타입	메소드(매개변수)	설명
Object	array()	버퍼가 래핑(wrap)한 배열을 리턴
int	arrayOffset()	버퍼의 첫번째 요소가 있는 내부 배열의 인덱스를 리턴
int	capacity()	버퍼의 전체 크기를 리턴.
Buffer	clear()	버퍼의 위치 속성을 초기화(position=0, limit=capacity)
Buffer	flip()	limit 을 position 으로, position 을 0 인덱스로 이동
boolean	hasArray()	버퍼가 래핑(wrap)한 배열을 가지고 있는지 여부
boolean	hasRemaining()	position 과 limit 사이에 요소가 있는지 여부(position<limit)
boolean	isDirect()	운영체제의 버퍼를 사용하는지 여부
boolean	isReadOnly()	버퍼가 읽기 전용인지 여부
int	limit()	limit 위치를 리턴
Buffer	limit(int newLimit)	newLimit 으로 limit 위치를 설정
Buffer	mark()	현재 위치를 mark 로 표시
int	position()	position 위치를 리턴
Buffer	position(int newPosition)	newPosition 으로 position 위치를 설정
int	remaining()	position 과 limit 사이의 요소의 개수
Buffer	reset()	position 을 mark 위치로 이동
Buffer	rewind()	position 을 0 인덱스로 이동

(2) 데이터를 읽고 저장하는 메소드

■ 데이터 읽기 get(...) vs. 데이터 저장 put(...)
- Buffer 추상 클래스에는 없고, 각 타입 별 하위 Buffer 클래스가 가지고 있다.

■ 상대적 vs. 절대적 메소드
- 상대적 메소드 : index 매개 변수가 없다. 호출하면 position 값은 증가된다.
- 절대적 메소드 : index 매개 변수가 있다. 호출하면 position 값은 증가되지 않는다.

(3) 버퍼 예외의 종류

■ 버퍼 예외 발생 주요 원인
- 버퍼가 다 찼을 때, 데이터를 저장하려는 경우
- 버퍼에서 더 이상 읽어올 데이터가 없을 때 데이터를 읽으려는 경우

■ 버퍼와 관련된 예외 클래스

예외	설명
BufferOverflowException	position 이 limit 에 도달했을 때 put() 호출하면 발생
BufferUnderflowException	position 이 limit 에 도달했을 때 get() 호출하면 발생
InvalidMarkException	mark 가 없는 상태에서 reset() 메소드를 호출하면 발생
ReadOnlyBufferException	읽기 전용 버퍼에서 put() 또는 compact() 메소드를 호출하면 발생

19.3.5 Buffer 변환

■ 채널이 데이터를 저장하고 읽는 버퍼는 모두 ByteBuffer

■ 프로그램 목적 맞게 ByteBuffer를 다른 기본 타입 버퍼로 변환 필요

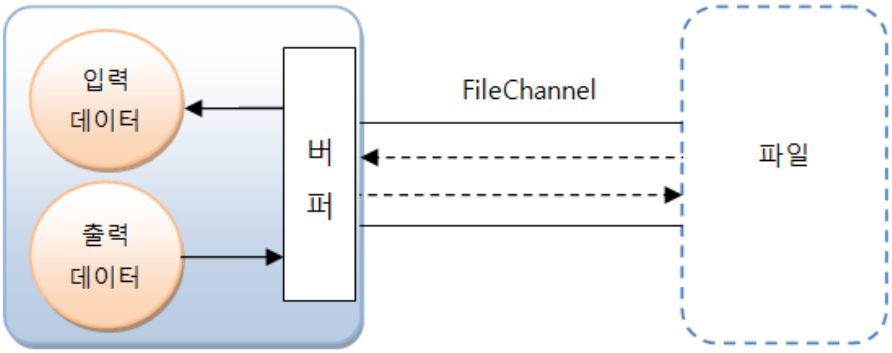
■ ByteBuffer <-> String

■ ByteBuffer <-> IntBuffer

19.4 파일 채널

■ 파일 읽기와 쓰기 가능하게 해주는 역할

■ 동기화 처리가 되어 있기 때문에 멀티 스레드 환경에서 사용해도 안전



19.4.1 FileChannel 생성과 닫기

■ 정적 메소드인 open()을 호출해서 얻을 수도 있지만, IO의 FileInputStream, FileOutputStream의 getChannel() 메소드를 호출해서 얻을 수도 있다.

■ StandardOpenOption의 다음 열거 상수를 나열해주면 된다.

열거 상수	설명
READ	읽기용으로 파일을 연다.
WRITE	쓰기용으로 파일을 연다.
CREATE	파일이 없다면 새 파일을 생성한다.
CREATE_NEW	새 파일을 만든다. 파일이 이미 있으면 예외와 함께 실패한다.
APPEND	파일 끝에 데이터를 추가한다(WRITE 나 CREATE 와 함께 사용됨)
DELETE_ON_CLOSE	스트림을 닫을 때 파일을 삭제한다(임시파일을 삭제할 때 사용)
TRUNCATE_EXISTING	파일을 0 바이트로 잘라낸다(WRITE 옵션과 함께 사용됨)

```
FileChannel fileChannel = FileChannel.open(
   Paths.get("C:/Temp/file.txt"),
   StandardOpenOption.CREATE_NEW,
   StandardOpenOption.WRITE
);

fileChannel.close();
```

19.4.2 파일 쓰기와 읽기

■ 파일에 바이트를 쓰려면 FileChannel의 write() 메소드를 호출하면 된다.

```
//파일 쓰기
```

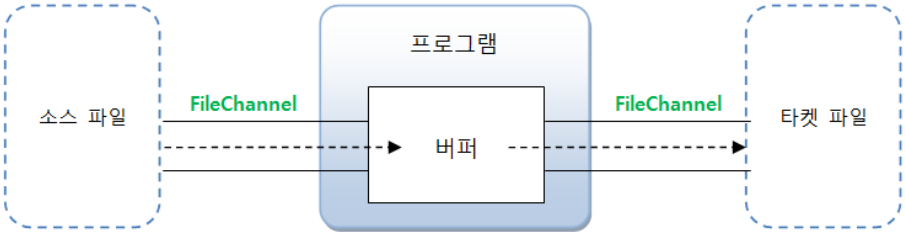
```
int bytesCount = fileChannel.write(ByteBuffer src);
//파일 읽기
int bytesCount = fileChannel.read(ByteBuffer dst);
```

[FileChannelWriteExample.java] 파일 생성 및 쓰기

```
package sec04.exam01_file_read_write;

import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.CharBuffer;
import java.nio.channels.FileChannel;
import java.nio.charset.Charset;
import java.nio.charset.CharsetEncoder;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;

public class FileChannelWriteExample {
	public static void main(String[] args) throws IOException {
		Path path = Paths.get("C:/Temp/file.txt");
		Files.createDirectories(path.getParent());

		FileChannel fileChannel = FileChannel.open(
			path, StandardOpenOption.CREATE, StandardOpenOption.WRITE);

		String data = "안녕하세요";
		Charset charset = Charset.defaultCharset();
		ByteBuffer byteBuffer = charset.encode(data);

		int byteCount = fileChannel.write(byteBuffer);
		System.out.println("file.txt : " + byteCount + " bytes written");

		fileChannel.close();
	}
}
```

19.4.3 파일 복사

■ 하나의 ByteBuffer를 사이에 두고 파일 읽기용 FileChannel과 파일 쓰기용 FileChannel이 읽기와 쓰기를 교대로 번갈아 수행하도록 하면 된다.



[FileCopyExample.java] 파일 복사

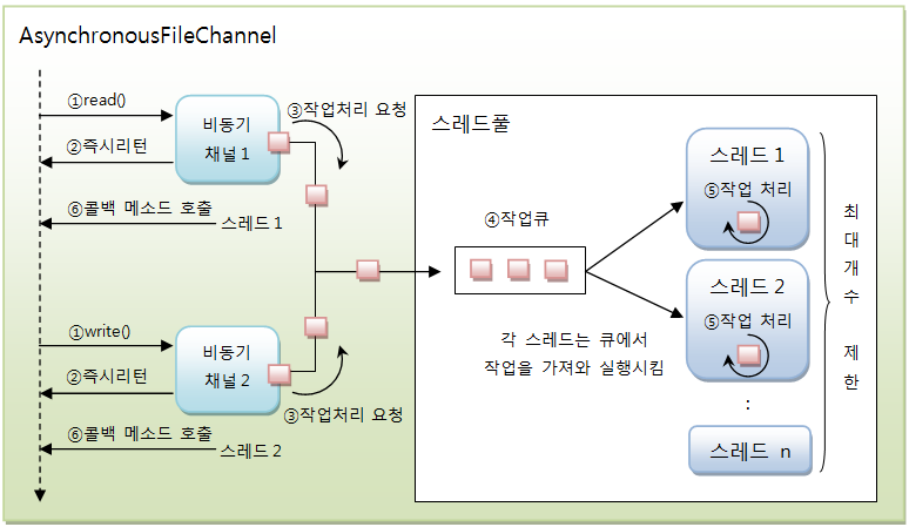
```
package sec04.exam02_file_copy;

import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
```

```java
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;

public class FileCopyExample {
	public static void main(String[] args) throws IOException {
		Path from = Paths.get("src/sec04/exam02_file_copy/house1.jpg");
		Path to = Paths.get("src/sec04/exam02_file_copy/house2.jpg");

		FileChannel fileChannel_from = FileChannel.open(
			from, StandardOpenOption.READ);

		FileChannel fileChannel_to = FileChannel.open(
			to, StandardOpenOption.CREATE, StandardOpenOption.WRITE);

		ByteBuffer buffer = ByteBuffer.allocateDirect(100);
		int byteCount;
		while(true) {
			buffer.clear();
			byteCount = fileChannel_from.read(buffer); // 읽기
			if(byteCount == -1) break;
			buffer.flip();
			fileChannel_to.write(buffer);  // 쓰기
		}

		fileChannel_from.close();
		fileChannel_to.close();
		System.out.println("파일 복사 성공");
	}
}
```

19.5 파일 비동기 채널

- FileChannel의 단점: read()와 write() 메소드는 작업하는 동안 블로킹
  - 블로킹 동안에 UI 갱신이나 이벤트 처리를 할 수 없음
  - 따라서 별도의 작업 스레드를 생성해서 이들 메소드를 호출해야
  - 동시에 처리해야 할 파일 수가 많다면 스레드 수 증가로 문제 유발 가능

- AsynchronousFileChannel: read()와 write() 메소드는 즉시 리턴
  - 이들 메소드는 스레드풀에게 작업 처리를 요청하고 즉시 리턴
  - 작업 스레드가 파일 입출력 완료 -> 콜백(callback) 메소드 자동 호출
  - 불특정 다수의 파일 및 대용량 파일의 입출력 작업 시 유리



19.5.1. AsynchronousFileChannel 생성과 닫기

```
// 스레드풀 생성
ExecutorService executorService = Executors.newFixedThreadPool(
   Runtime.getRuntime().availableProcessors()
);

// 비동기 파일 채널 생성
AsynchronousFileChannel fileChannel = AsynchronousFileChannel.open {
   Paths.get("C:/Temp/file.txt"),
   EnumSet.of(StandardOpenOption.CREATE, StandardOpenOption.WRITE),
   executorService
}

// 채널 닫기
fileChannel.close();
```

19.5.2. 파일 읽기와 쓰기

■ 매개변수
- dst, src: 읽거나 쓰기 위한 ByteBuffer
- position: 파일에서 읽을 위치이거나 쓸 위치
- attachment: 콜백 메소드로 전달할 첨부 객체
- handler: CompletionHandler<Integer, A> 구현 객체

■ CompletionHandler<Integer, A>
- Integer: 입출력 작업 처리 후 결과 타입 (결과값은 읽거나 쓴 바이트 수) (고정)
- A: 첨부 객체 타입으로 첨부 객체가 필요 없다면 Void 지정(개발자 지정)

```
read(ByteBuffer dst, long position, A attachment, Completionhandler<Integer, A> handler);
write(ByteBuffer src, long position, A attachment, Completionhandler<Integer, A> handler);

new CompletionHandler<Integer, A> {
   @Override
```

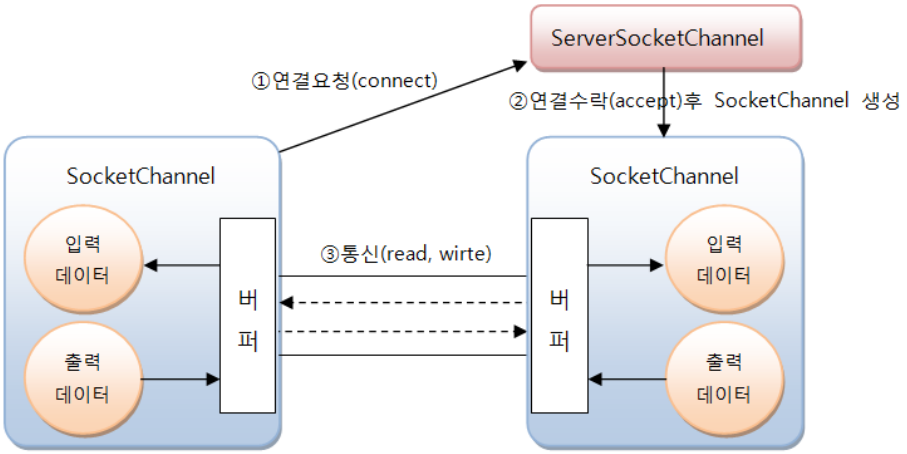
```
    public void completed(Integer result, A attachment) {...}
    @Override
    public void failed(Throwable exec, A attachment) {...}
```

19.6 TCP 블로킹 채널

- NIO를 이용해서 TCP 서버/클라이언트 애플리케이션을 개발하려면 블로킹, 넌블로킹, 비동기 구현 방식 중에서 하나를 결정해야 한다.

19.6.1. 서버소켓 채널과 소켓 채널의 용도

- ServerSocketChannel은 클라이언트 SocketChannel의 연결 요청을 수락하고 통신용 SocketChannel을 생성한다.



19.6.2. 서버소켓 채널 생성과 연결 수락

```
// ServerSocketChannel 생성
ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
serverSocketChannel.configureBlocking(true);
serverSocketChannel.bind(new InetSocketAddress(5001));

// 연결 수락
ServerChannel socketChannel = serverSocketChannel.accept();

InetSocketAddress socketAddress = (InetSocketAddress) socketChannel.getRemoteAddress();

// 닫기
serverSocketChannel.close();
```

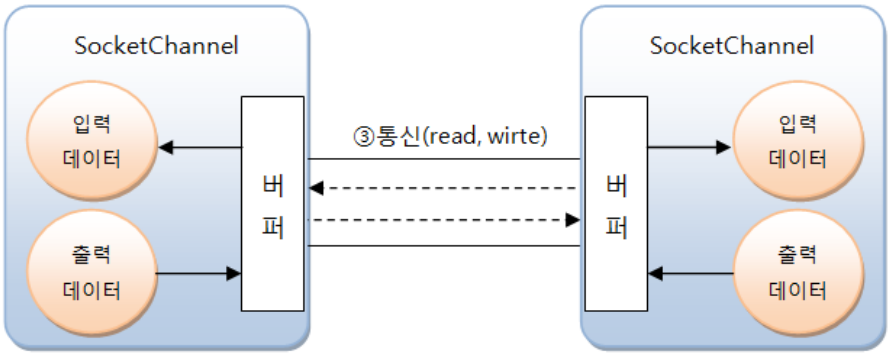
19.6.3. 소켓 채널 생성과 연결 요청

- SocketChannel 생성과 연결 요청
  - 클라이언트가 서버에 연결 요청할 때 쓰이는 소켓

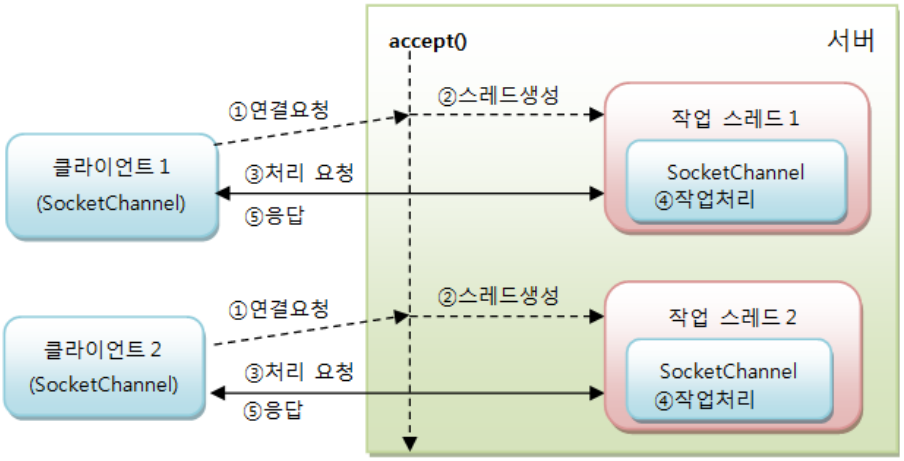
- 닫기
  - 클라이언트가 종료되거나, 필요에 따라 연결 끊을 때 Close()

- 서버가 열려있어야 클라이언트 통신 가능

19.6.4. 소켓 채널 데이터 통신



19.6.5. 스레드 병렬 처리



19.6.6. 채팅 서버 구현

(1) 서버 클래스 구조

- 다음은 서버 클래스의 구조를 보여준다.

ServerExample.java] 채팅 서버

```
01    package sec06.exam03_chatting;
02
03    import java.io.IOException;
04    import java.net.InetSocketAddress;
05    import java.nio.ByteBuffer;
06    import java.nio.channels.ServerSocketChannel;
```

```
07   import java.nio.channels.SocketChannel;
08   import java.nio.charset.Charset;
09   import java.util.Iterator;
10   import java.util.List;
11   import java.util.Vector;
12   import java.util.concurrent.ExecutorService;
13   import java.util.concurrent.Executors;
14
15   import javafx.application.Application;
16   import javafx.application.Platform;
17   import javafx.geometry.Insets;
18   import javafx.scene.Scene;
19   import javafx.scene.control.Button;
20   import javafx.scene.control.TextArea;
21   import javafx.scene.layout.BorderPane;
22   import javafx.stage.Stage;
23
24   public class ServerExample extends Application {
25             ExecutorService executorService;
26             ServerSocketChannel serverSocketChannel;
27             List<Client> connections = new Vector<Client>();
28
29             void startServer() {
30                       // 서버 시작 코드
31             }
32
33             void stopServer() {
34                       // 서버 종료 코드
35             }
36
37             class Client {
38                       // 데이터 통신 코드
39             }
40
41             ////////////////////////////////////////////////////
42             // UI 생성 코드
43   }
```

(2) startServer() 메소드

- [start] 버튼을 클릭하면 startServer() 메소드가 실행되는데, startServer() 메소드에서는 ExecutorService 생성, ServerSocketChannel 생성 및 포트 바인딩, 연결 수락 코드가 필요하다.

ServerExample.java] startServer() 메소드 - 서버 시작 코드

```
01             // 생략
02             void startServer() {
03                       executorService = Executors.newFixedThreadPool(
04                                 Runtime.getRuntime().availableProcessors()
05                      );
06
07                       try {
08                                 serverSocketChannel = ServerSocketChannel.open();
09                                 serverSocketChannel.configureBlocking(true);
10                                 serverSocketChannel.bind(new InetSocketAddress(5001));
11                       } catch(Exception e) {
12                                 if(serverSocketChannel.isOpen()) { stopServer(); }
13                                 return;
14                       }
15
```

```
16                      Runnable runnable = new Runnable() {
17                              @Override
18                              public void run() {
19                                      Platform.runLater(()->{
20                                              displayText("[서버 시작]");
21                                              btnStartStop.setText("stop");
22                                      });
23                                      while(true) {
24                                              try {
25                                                      SocketChannel        socketChannel        =
26   serverSocketChannel.accept();
27                                                      String   message   =   "[연결   수락:   "   +
28   socketChannel.getRemoteAddress()  + ": " + Thread.currentThread().getName() + "]";
29                                                      Platform.runLater(()->displayText(message));
30
31                                                      Client client = new Client(socketChannel);
32                                                      connections.add(client);
33
34                                                      Platform.runLater(()->displayText("[연결  개
35   수: " + connections.size() + "]"));
36                                              } catch (Exception e) {
37                                                      if(serverSocketChannel.isOpen())
38   { stopServer(); }
39                                                      break;
40                                              }
41                                      }
42                              }
43                      };
44                      executorService.submit(runnable);
45              }
46              // 생략
```

(3) stopServer() 메소드

- [stop] 버튼을 클릭하면 stopServer() 메소드가 실행되는데, stopServer() 메소드에는 연결된 모든 SocketChannel 닫기, ServerSocketChannel 닫기, ExecutorService 종료 코드가 필요하다.

ServerExample.java] stopServer() 메소드 - 서버 종료 코드

```
01              // 생략
02              void stopServer() {
03                      try {
04                              Iterator<Client> iterator = connections.iterator();
05                              while(iterator.hasNext()) {
06                                      Client client = iterator.next();
07                                      client.socketChannel.close();
08                                      iterator.remove();
09                              }
10                              if(serverSocketChannel!=null && serverSocketChannel.isOpen()) {
11                                      serverSocketChannel.close();
12                              }
13                              if(executorService!=null && !executorService.isShutdown()) {
14                                      executorService.shutdown();
15                              }
16                              Platform.runLater(()->{
17                                      displayText("[서버 멈춤]");
18                                      btnStartStop.setText("start");
19                              });
20                      } catch (Exception e) {}
21              }
```

```
22            // 생략
```

(4) Client 클래스

- 클라이언트별로 고유한 데이터를 저장할 필요도 있기 때문에 Client 클래스를 작성하고, 연락 수락 시 마다 Client 인스턴스를 생성해서 관리하는 것이 좋다.

ServerExample.java] Class Client - 데이터 통신 코드

```
01            // 생략
02            class Client {
03                      SocketChannel socketChannel;
04
05                      Client(SocketChannel socketChannel) {
06                              this.socketChannel = socketChannel;
07                              receive();
08                      }
09
10                      void receive() {
11                              Runnable runnable = new Runnable() {
12                                      @Override
13                                      public void run() {
14                                              while(true) {
15                                                      try {
16                                                              ByteBuffer      byteBuffer      =
17  ByteBuffer.allocate(100);
18
19                                                              //클라이언트가 비정상 종료를 했을
20  경우 IOException 발생
21                                                              int        readByteCount        =
22  socketChannel.read(byteBuffer);
23
24                                                              //클라이언트가          정상적으로
25  SocketChannel의 close()를 호출했을 경우
26                                                              if(readByteCount == -1) {
27                                                                      throw new IOException();
28                                                              }
29
30                                                              String message = "[요청 처리: " +
31  socketChannel.getRemoteAddress() + ": " + Thread.currentThread().getName() + "]";
32                                                              Platform.runLater(()->
33  displayText(message));
34
35                                                              byteBuffer.flip();
36                                                              Charset         charset         =
37  Charset.forName("UTF-8");
38                                                              String           data           =
39  charset.decode(byteBuffer).toString();
40
41                                                              for(Client client : connections) {
42                                                                      client.send(data);
43                                                              }
44                                                      } catch(Exception e) {
45                                                              try {
46
47              connections.remove(Client.this);
48                                                                      String message = "[클라
49  이언트 통신 안됨: " + socketChannel.getRemoteAddress() + ": " + Thread.currentThread().getName() + "]";
50                                                                      Platform.runLater(()->
51  displayText(message));
52                                                                      socketChannel.close();
```

```
53                                                                  } catch (IOException e2) {}
54                                                                  break;
55                                                          }
56                                                  }
57                                          }
58                                  };
59                                  executorService.submit(runnable);
60                          }
61
62                          void send(String data) {
63                                  Runnable runnable = new Runnable() {
64                                          @Override
65                                          public void run() {
66                                                  try {
67                                                          Charset charset = Charset.forName("UTF-8");
68                                                          ByteBuffer           byteBuffer           =
69  charset.encode(data);
70                                                          socketChannel.write(byteBuffer);
71                                                  } catch(Exception e) {
72                                                          try {
73                                                                  String message = "[클라이언트 통신
74  안됨: " + socketChannel.getRemoteAddress() + ": " + Thread.currentThread().getName() + "]";
75                                                                  Platform.runLater(()->
76  displayText(message));
77                                                                  connections.remove(Client.this);
78                                                                  socketChannel.close();
79                                                          } catch (IOException e2) {}
80                                                  }
81                                          }
82                                  };
83                                  executorService.submit(runnable);
84                          }
85                  }
86                  // 생략
```

(5) UI 생성 코드

- 다음은 ServerExample UI 생성 코드를 보여준다. 프로그램적 레이아웃을 이용해서 컴포넌트를 배치했다.

ServerExample.java] UI 생성 코드

```
01                  // 생략
02                  ////////////////////////////////////////////////////
03                  TextArea txtDisplay;
04                  Button btnStartStop;
05
06                  @Override
07                  public void start(Stage primaryStage) throws Exception {
08                          BorderPane root = new BorderPane();
09                          root.setPrefSize(500, 300);
10
11                          txtDisplay = new TextArea();
12                          txtDisplay.setEditable(false);
13                          BorderPane.setMargin(txtDisplay, new Insets(0,0,2,0));
14                          root.setCenter(txtDisplay);
15
16                          btnStartStop = new Button("start");
17                          btnStartStop.setPrefHeight(30);
18                          btnStartStop.setMaxWidth(Double.MAX_VALUE);
19                          btnStartStop.setOnAction(e->{
```

```
20                                  if(btnStartStop.getText().equals("start")) {
21                                          startServer();
22                                  } else if(btnStartStop.getText().equals("stop")){
23                                          stopServer();
24                                  }
25                          });
26                          root.setBottom(btnStartStop);
27
28                          Scene scene = new Scene(root);
29                          scene.getStylesheets().add(getClass().getResource("app.css").toString());
30                          primaryStage.setScene(scene);
31                          primaryStage.setTitle("Server");
32                          primaryStage.setOnCloseRequest(event->stopServer());
33                          primaryStage.show();
34                  }
35
36                  void displayText(String text) {
37                          txtDisplay.appendText(text + "\n");
38                  }
39
40                  public static void main(String[] args) {
41                          launch(args);
42                  }
43                  // 생략
```

19.6.7. 채팅 클라이언트 구현

(1) 클라이언트 클래스 구조

- 다음은 클라이언트 클래스의 구조를 보여준다.

[ClientExample.java]

```
01    package sec06.exam03_chatting;
02
03    import java.io.IOException;
04    import java.net.InetSocketAddress;
05    import java.nio.ByteBuffer;
06    import java.nio.channels.SocketChannel;
07    import java.nio.charset.Charset;
08
09    import javafx.application.Application;
10    import javafx.application.Platform;
11    import javafx.geometry.Insets;
12    import javafx.scene.Scene;
13    import javafx.scene.control.Button;
14    import javafx.scene.control.TextArea;
15    import javafx.scene.control.TextField;
16    import javafx.scene.layout.BorderPane;
17    import javafx.stage.Stage;
18
19    public class ClientExample extends Application {
20              SocketChannel socketChannel;
21
22              void startClient() {
23                        // 연결 시작 코드
24              }
25
26              void stopClient() {
27                        // 연결 끊기 코드
```

```
28            }
29
30            void receive() {
31                    // 데이터 받기 코드
32            }
33
34            void send(String data) {
35                    // 데이터 전송 코드
36            }
37
38            ////////////////////////////////////////////////////
39            // UI 생성 코드
40    }
```

(2) startClient() 메소드

- [start] 버튼을 클릭하면 startClient() 메소드가 실행되는데, startClient() 메소드에는 Socket 생성 및 연결 요청 코드가 있어야 한다.

[ClientExample.java]

```
01            // 생략
02            void startClient() {
03                    Thread thread = new Thread() {
04                            @Override
05                            public void run() {
06                                    try {
07                                            socketChannel = SocketChannel.open();
08                                            socketChannel.configureBlocking(true);
09                                            socketChannel.connect(new
10    InetSocketAddress("localhost", 5001));
11                                            Platform.runLater(()->{
12                                                    try {
13                                                            displayText("[연결  완료:  "    +
14    socketChannel.getRemoteAddress() + "]");
15                                                            btnConn.setText("stop");
16                                                            btnSend.setDisable(false);
17                                                    } catch (Exception e) {}
18                                            });
19                                    } catch(Exception e) {
20                                            Platform.runLater(()->displayText("[서버    통신    안
21    됨]"));
22                                            if(socketChannel.isOpen()) { stopClient(); }
23                                            return;
24                                    }
25                                    receive();
26                            }
27                    };
28                    thread.start();
29            }
30            // 생략
```

(3) stopClient() 메소드

- [stop] 버튼을 클릭하거나 서버 통신이 안 될 경우 stopClient() 메소드가 실행되는데, stopClinet() 메소드에는 SocketChannel을 닫는 close() 메소드 호출 코드가 있어야 한다.

[ClientExample.java]

```
01              // 생략
02              void stopClient() {
03                        try {
04                                  Platform.runLater(()->{
05                                            displayText("[연결 끊음]");
06                                            btnConn.setText("start");
07                                            btnSend.setDisable(true);
08                                  });
09                                  if(socketChannel!=null && socketChannel.isOpen()) {
10                                            socketChannel.close();
11                                  }
12                        } catch (IOException e) {}
13              }
14              // 생략
```

(4) receive() 메소드

- receive() 메소드는 서버에서 보낸 데이터를 받는 역할을 한다. 이 메소드는 startClient() 에서 생성한 작업 스레드상에서 호출된다.

[ClientExample.java]

```
01              // 생략
02              void receive() {
03                        while(true) {
04                                  try {
05                                            ByteBuffer byteBuffer = ByteBuffer.allocate(100);
06
07                                            //서버가 비정상적으로 종료했을 경우 IOException 발생
08                                            int readByteCount = socketChannel.read(byteBuffer);
09
10                                            //서버가 정상적으로 Socket의 close()를 호출했을 경우
11                                            if(readByteCount == -1) {
12                                                      throw new IOException();
13                                            }
14
15                                            byteBuffer.flip();
16                                            Charset charset = Charset.forName("UTF-8");
17                                            String data = charset.decode(byteBuffer).toString();
18
19                                            Platform.runLater(()->displayText("[받기 완료] "  + data));
20                                  } catch (Exception e) {
21                                            Platform.runLater(()->displayText("[서버 통신 안됨]"));
22                                            stopClient();
23                                            break;
24                                  }
25                        }
26              }
27              // 생략
```

(5) send(String data) 메소드

- 사용자가 메시지를 입력하고 [send] 버튼을 클릭하면 메시지를 매개값으로 해서 send(String data) 메소드가 호출된다. 이 메소드는 메시지를 서버로 보내는 역할을 한다.

```
            // 생략
            void send(String data) {
                    Thread thread = new Thread() {
                            @Override
                            public void run() {
                                    try {
                                            Charset charset = Charset.forName("UTF-8");
                                            ByteBuffer byteBuffer = charset.encode(data);
                                            socketChannel.write(byteBuffer);
                                            Platform.runLater(()->displayText("[보내기 완료]"));
                                    } catch(Exception e) {
                                            Platform.runLater(()->displayText("[서버    통신    안
됨]"));
                                            stopClient();
                                    }
                            }
                    };
                    thread.start();
            }
            // 생략
```

(6) UI 생성 코드

- 다음은 ClientExample UI 생성 코드를 보여준다.

[ClientExample.java]

```
            // 생략
            ////////////////////////////////////////////////////
            TextArea txtDisplay;
            TextField txtInput;
            Button btnConn, btnSend;

            @Override
            public void start(Stage primaryStage) throws Exception {
                    BorderPane root = new BorderPane();
                    root.setPrefSize(500, 300);

                    txtDisplay = new TextArea();
                    txtDisplay.setEditable(false);
                    BorderPane.setMargin(txtDisplay, new Insets(0,0,2,0));
                    root.setCenter(txtDisplay);

                    BorderPane bottom = new BorderPane();
                            txtInput = new TextField();
                            txtInput.setPrefSize(60, 30);
                            BorderPane.setMargin(txtInput, new Insets(0,1,1,1));

                            btnConn = new Button("start");
                            btnConn.setPrefSize(60, 30);
                            btnConn.setOnAction(e->{
                                    if(btnConn.getText().equals("start")) {
                                            startClient();
                                    } else if(btnConn.getText().equals("stop")){
                                            stopClient();
                                    }
                            });

                            btnSend = new Button("send");
                            btnSend.setPrefSize(60, 30);
```

```
34                                  btnSend.setDisable(true);
35                                  btnSend.setOnAction(e->send(txtInput.getText()));
36
37                                  bottom.setCenter(txtInput);
38                                  bottom.setLeft(btnConn);
39                                  bottom.setRight(btnSend);
40                          root.setBottom(bottom);
41
42                          Scene scene = new Scene(root);
43                          scene.getStylesheets().add(getClass().getResource("app.css").toString());
44                          primaryStage.setScene(scene);
45                          primaryStage.setTitle("Client");
46                          primaryStage.setOnCloseRequest(event->stopClient());
47                          primaryStage.show();
48                  }
49
50                  void displayText(String text) {
51                          txtDisplay.appendText(text + "\n");
52                  }
53
54                  public static void main(String[] args) {
55                          launch(args);
56                  }
57                  // 생략
```

19.6.8. 블로킹과 인터럽트

- IO 소켓에서는 입출력 스트림에서 작업스레드가 블로킹 된 경우
  - 다른 스레드가 작업 스레드의 인터럽트 메소드 호출해도 블로킹이 풀리지 않음

- NIO 소켓 채널의 경우
  - 인터럽트 만으로도 소켓채널이 닫히면서 블로킹 풀림

19.7 TCP 넌블로킹 채널

19.7.1. 넌블로킹 방식의 특징

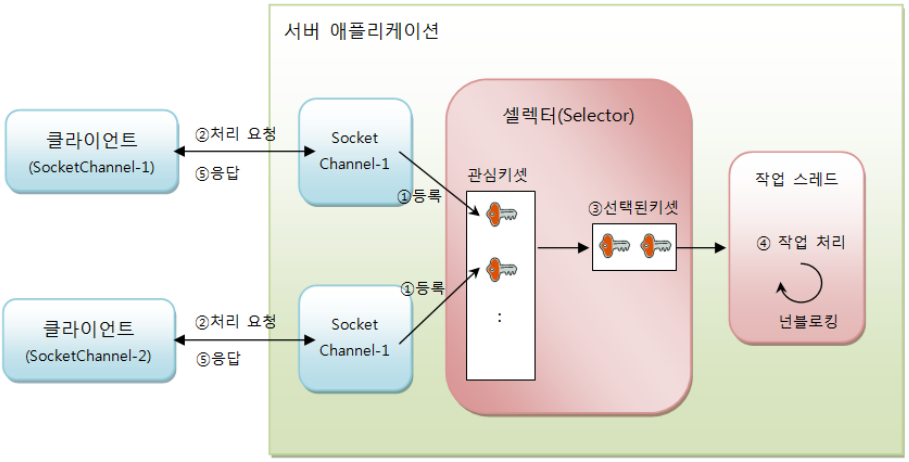
- connect(), accept(), read(), write() 메소드는 블로킹 없이 즉시 리턴
  - 작업 처리 준비가 된 상태에서 메소드 실행할 것
  - 작업 처리 준비가 된 채널만 선택해 처리

- 셀렉터가 작업 처리 준비된 채널 선택
  - 넌블로킹 채널은 이벤트 리스너 역할 하는 셀렉터(Selector) 사용
  - 채널이 작업 처리 필요할 경우 셀렉터에 통보
  - 셀렉터는 통보한 채널 선택

- 멀티 채널 작업을 싱글 스레드에서 처리 가능
  - 작업 스레드가 블로킹되지 않음 -> 셀렉터가 선택한 채널들을 싱글 스레드에서 모두 처리 가능
  - 스레드 풀 사용할 경우, 적은 수의 스레드로 많은 양의 작업 처리

- 셀렉터(Selector)의 동작 원리

- 채널은 자신의 작업 유형을 키(SelectionKey)로 생성
- 셀렉터의 관심 키셋(interest-set)에 키 등록
- 셀렉터는 작업 처리 준비가 된 키를 선택
- 선택된 키셋에 별도로 저장
- 작업 스레드는 선택된 키셋에서 키를 하나씩 꺼냄



19.7.2. 셀렉터 생성과 등록

■ 셀렉터 생성 - Open() 메소드 호출해 생성
- Exception 발생 가능하므로 예외처리 필요

■ 넌블로킹 채널 생성 - 하위 클래스도 넌블로킹이어야.

■ 셀렉터 등록
- 첫 번째 매개값은 Selector
- 두 번째 매개값은 작업 유형별 SelectionKey의 상수

19.7.3. 선택된 키셋

■ Selector의 select() 메소드
- 관심 키셋의 SelectionKey로부터 작업 처리 준비가 되었다는 통보 올 때까지 블로킹
  - 최소한 하나의 SelectionKey로부터 작업 처리 준비가 되었다는 통보가 오면 리턴
  - 리턴값은 통보를 해온 SelectionKey의 수

■ select() 메소드 종류

리턴타입	메소드명(매개변수)	설명
int	select()	최소한 하나의 채널이 작업 처리 준비가 될때까지 블로킹된다.
int	select(long timeout)	select()와 동일한데, 주어진 시간(밀리세컨)동안만 블로킹된다.
int	selectNow()	작업을 처리할 준비가 된 채널만 선택하고 즉시 리턴된다.

19.7.4. 채널 작업 처리

■ 선택된 키셋에서 SelectionKey를 하나씩 꺼내어 작업 유형별 채널 작업 처리

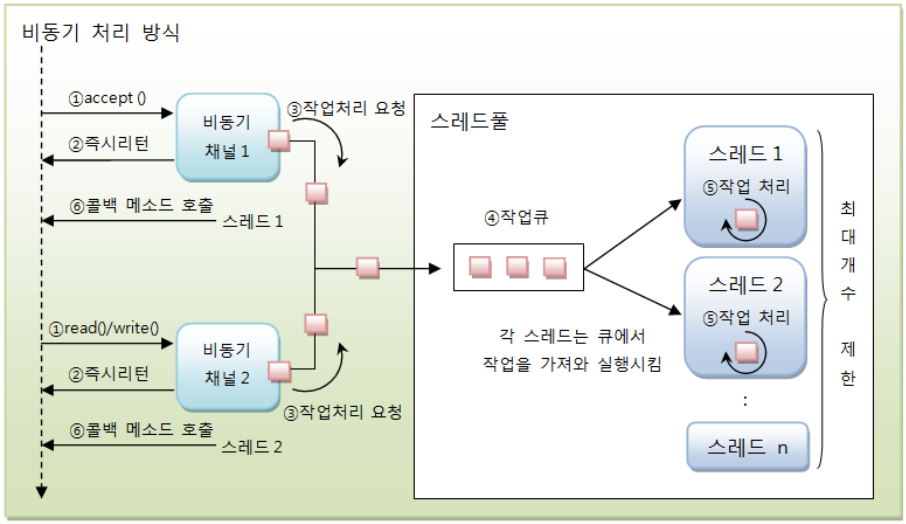
~~19.7.5. 채팅 서버 구현~~

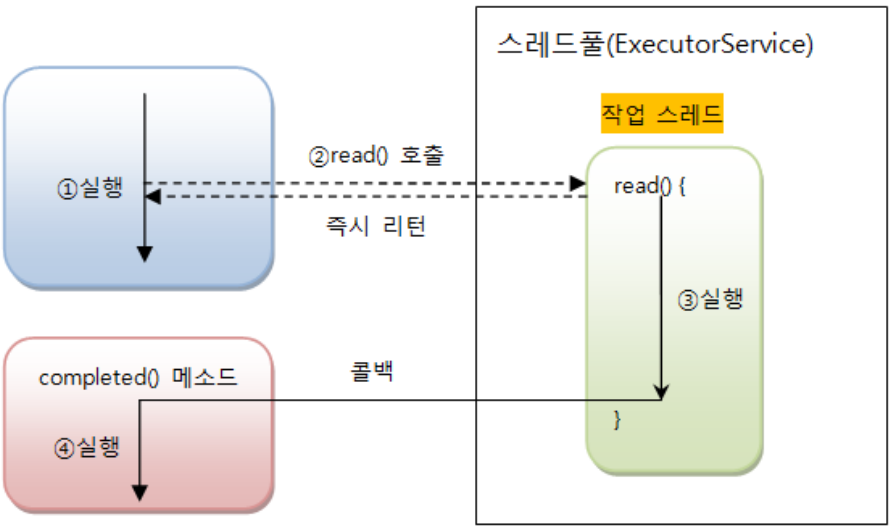
~~19.7.6. 채팅 클라이언트 구현~~

19.8 TCP 비동기 채널

19.8.1. 비동기 채널의 특징

■ connect(), accept(), read(), write()를 호출하면 즉시 리턴된다.
- 실질적 입출력 작업 처리는 스레드 풀의 스레드가 담당한다.
- 스레드가 작업 처리 완료하면 콜백 메소드가 자동 호출된다.

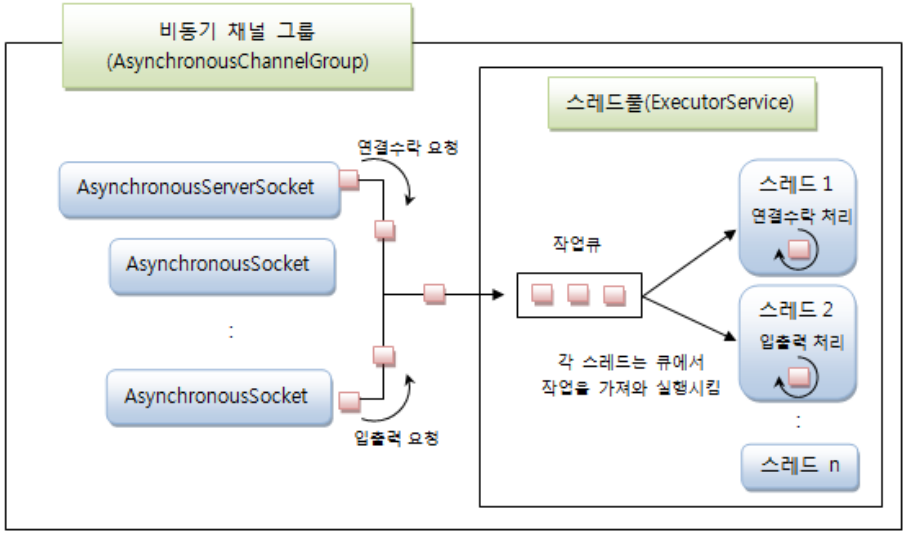




19.8.2. 비동기 채널 그룹

■ 비동기 채널 그룹(AsynchronousChannelGroup)
- 같은 스레드풀 공유하는 비동기 채널들의 묶음

- 하나의 스레드풀을 사용한다면 모든 비동기 채널은 같은 채널 그룹



■ 비동기 채널 그룹 생성
- 비동기 채널 생성할 때 채널 그룹 지정하지 않으면 기본 비동기 채널 그룹
- 기본 비동기 채널 그룹은 내부적으로 생성되는 스레드풀 이용

■ 비동기 채널 그룹 종료
- shutdown()
  - 비동기 채널 그룹을 종료하겠다는 의사만 전달
  - 즉시 비동기 채널 그룹을 종료하지 않음
  - 비동기 채널 그룹에 포함된 모든 비동기 채널이 닫히면 종료
  - 새로운 비동기 채널을 포함시키려고 하면 ShutdownChannelGroupException이 발생
- shutdownNow()
  - 강제적으로 비동기 채널 그룹에 포함된 모든 비동기 채널 닫고 비동기 채널 그룹을 종료
  - 완료 콜백 실행하고 있는 스레드는 종료되거나 인터럽트 X

```
AsynchromousChannelGroup channelGroup = AsynchronousChannelGroup.withFixedThreadPool(
   Runtime.getRuntime().availableProcessors(),  // 최대스레드 수
   Executors.defaultThreadFactory()
);

channelGroup.shutdown();
channelGroup.shutdownNow();
```

19.8.3. 비동기 서버소켓 채널

1. 비동기 채널 그룹을 생성한다.
2. 새로 생성한 비동기 채널 그룹에 포함되는 비동기 서버 채널 생성한다.
3. 포트 바인딩한다.
4. 연결 수락 작업을 스레드풀을 이용해서 비동기로 처리한다.
5. 더 이상 소켓 채널이 사용되지 않으면 닫는다.

```
// 1. 비동기 채널 그룹 생성
```

```
AsynchromousChannelGroup channelGroup = AsynchronousChannelGroup.withFixedThreadPool(
   Runtime.getRuntime().availableProcessors(),
   Executors.defaultThreadFactory()
);
channelGroup.shutdown();
channelGroup.shutdownNow();

// 2. 비동기 서버 채널 생성
AsynchronousServerSocketChannel asynchronousServerSocketChannel =
AsynchronousServerSocketChannel.open(channelGroup);

// 3. 포트 바인딩
asynchronousServerSocketChannel.bind(new InetSocketAddress(5001));

// 4. 연결 수락 작업
asynchronousServerSocketChannel.accept(null,
   new CompletionHandler<AsynchronousScoketChannel, Void>() {
   @Override
   public void completed(AsynchronousServerSocketChannel asynchronousServerSocketChannel,
                         Void attachemnet) {
      // 연결 수락 후 실행할 코드
      asynchronousServerSocketChannel.accept(null, this);  // accept() 재호출
   }
   @Overrider
   public void failed(Throwable exc, Void attachment) {
      // 연결 수락 실패 시 실행할 코드
   }
};

// 5. 닫기
asynchronousServerSocketChannel.close();
```

19.8.4. 비동기 소켓 채널

■ 클라이언트와 서버 연결 후의 통신

```
asynchronousSocketChannel.connect(new InetSocketAddress("localhost", 5001), null,
   new CompletionHandler<Void, Void>() {
   @Override
   public void completed(Void result, Void attachment) {
      // 연결 성공후 실행할 코드
   }
   @Overrider
   public void failed(Throwable e, Void attachment) {
      // 연결 실패후 실행할 코드
   }
});
```

19.8.5. 비동기 소켓 채널 데이터 통신

■ 클라이언트와 서버가 연결되면 양쪽 AsynchronousServerSocketChannel의 read()와 write() 메소드로 데이터 통신을 할 수 있다.

```
asynchronousSocketChannel.read(byteBuffer, attachment,
   new CompletionHandler<Integer, A>() {
```

```
   @Overrider
   public void completed(Integer result, A attachment) {
      // 받은 데이터를 처리하는 코드
      asynchronousSocketChannel.read(byteBuffer, attachment, this);  // read() 재호출
   }
   @Overrider
   public void failed(Throwable exc, A attachment) {
      // 실패된 경우 실행할 코드
   }
});

asynchronousSocketChannel.write(byteBuffer, attachment,
   new CompletionHandler<Integer, A>() {
   @Overrider
   public void completed(Integer result, A attachment) {
      // 성공한 경우 실행할 코드
   }
   @Overrider
   public void failed(Throwable exc, A attachment) {
      // 실패된 경우 실행할 코드
   }
});
```

19.8.6. 채팅 서버 구현

- 비동기 서버소켓채널과 비동기 소켓채널 사용법 이해
- 서버에는 다수의 클라이언트가 붙는다는 것을 전제로 작업
- UI의 경우는 TCP 동기채널의 작동과 동일

(1) 서버 클래스 구조

- 다음은 서버 클래스의 구조를 보여준다.

ServerExample.java] 채팅 서버

```
01    package sec08.exam01_asynchronous_tcpchannel;
02
03    import java.io.IOException;
04    import java.net.InetSocketAddress;
05    import java.nio.ByteBuffer;
06    import java.nio.channels.AsynchronousChannelGroup;
07    import java.nio.channels.AsynchronousServerSocketChannel;
08    import java.nio.channels.AsynchronousSocketChannel;
09    import java.nio.channels.CompletionHandler;
10    import java.nio.charset.Charset;
11    import java.util.List;
12    import java.util.Vector;
13    import java.util.concurrent.Executors;
14
15    import javafx.application.Application;
16    import javafx.application.Platform;
17    import javafx.geometry.Insets;
18    import javafx.scene.Scene;
19    import javafx.scene.control.Button;
20    import javafx.scene.control.TextArea;
21    import javafx.scene.layout.BorderPane;
```

```
22  import javafx.stage.Stage;
23
24  public class ServerExample extends Application {
25            AsynchronousChannelGroup channelGroup;
26            AsynchronousServerSocketChannel serverSocketChannel;
27            List<Client> connections = new Vector<Client>();
28
29            void startServer() {
30                      // 서버 시작 코드
31            }
32
33
34            void stopServer() {
35                      // 서버 종료 코드
36            }
37
38            class Client {
39                      // 데이터 통신 코드
40            }
41
42            ////////////////////////////////////////
43            // UI 생성 코드
44  }
```

(2) startServer() 메소드

ServerExample.java] startServer() 메소드

```
01  // 생략
02            void startServer() {
03                      try {
04                                channelGroup = AsynchronousChannelGroup.withFixedThreadPool(
05                                          Runtime.getRuntime().availableProcessors(),
06                                          Executors.defaultThreadFactory()
07                                );
08                                serverSocketChannel = AsynchronousServerSocketChannel.open(channelGroup);
09                                serverSocketChannel.bind(new InetSocketAddress(5001));
10                      } catch(Exception e) {
11                                if(serverSocketChannel.isOpen()) { stopServer(); }
12                                return;
13                      }
14
15                      Platform.runLater(()->{
16                                displayText("[서버 시작]");
17                                btnStartStop.setText("stop");
18                      });
19
20                      serverSocketChannel.accept(null,  new  CompletionHandler<AsynchronousSocketChannel,
21  Void>() {
22                                @Override
23                                public  void  completed(AsynchronousSocketChannel  socketChannel,  Void
24  attachment) {
25                                          try {
26                                                    String    message    =    "[연결    수락:    "    +
27  socketChannel.getRemoteAddress()  + ": " + Thread.currentThread().getName() + "]";
28                                                    Platform.runLater(()->displayText(message));
29                                          } catch (IOException e) {}
30
31                                          Client client = new Client(socketChannel);
32                                          connections.add(client);
33                                          Platform.runLater(()->displayText("[연결      개수:      "      +
34  connections.size() + "]"));
```

```
35
36                                      serverSocketChannel.accept(null, this);
37                              }
38                              @Override
39                              public void failed(Throwable exc, Void attachment) {
40                                      if(serverSocketChannel.isOpen()) { stopServer(); }
41                              }
42                      });
43              }
44      // 생략
```

(3) stopServer() 메소드

ServerExample.java] 채팅 서버

```
01      // 생략
02              void stopServer() {
03                      try {
04                              connections.clear();
05                              if(channelGroup!=null && !channelGroup.isShutdown()) {
06                                      channelGroup.shutdownNow();
07                              }
08                              Platform.runLater(()->{
09                                      displayText("[서버 멈춤]");
10                                      btnStartStop.setText("start");
11                              });
12                      } catch (Exception e) {}
13              }
14      // 생략
```

(4) Client 클래스

ServerExample.java] 채팅 서버

```
01      // 생략
02              class Client {
03                      AsynchronousSocketChannel socketChannel;
04
05                      Client(AsynchronousSocketChannel socketChannel) {
06                              this.socketChannel = socketChannel;
07                              receive();
08                      }
09
10                      void receive() {
11                              ByteBuffer byteBuffer = ByteBuffer.allocate(100);
12                              socketChannel.read(byteBuffer, byteBuffer, new CompletionHandler<Integer,
13      ByteBuffer>() {
14                                      @Override
15                                      public void completed(Integer result, ByteBuffer attachment) {
16                                              try {
17                                                      String message = "[요청 처리: " +
18      socketChannel.getRemoteAddress() + ": " + Thread.currentThread().getName() + "]";
19                                                      Platform.runLater(()->displayText(message));
20
21                                                      attachment.flip();
22                                                      Charset charset = Charset.forName("utf-8");
23                                                      String data =
24      charset.decode(attachment).toString();
25
```

```
                                                            for(Client client : connections) {
                                                                    client.send(data);
                                                            }

                                                            ByteBuffer            byteBuffer            =
ByteBuffer.allocate(100);
                                                            socketChannel.read(byteBuffer,   byteBuffer,
this);
                                                    } catch(Exception e) {}
                                            }
                                            @Override
                                            public void failed(Throwable exc, ByteBuffer attachment) {
                                                    try {
                                                            String message = "[클라이언트 통신 안됨: " +
socketChannel.getRemoteAddress() + ": " + Thread.currentThread().getName() + "]";
                                                            Platform.runLater(()->displayText(message));
                                                            connections.remove(Client.this);
                                                            socketChannel.close();
                                                    } catch (IOException e) {}
                                            }
                                    });
                            }

                            void send(String data) {
                                    Charset charset = Charset.forName("utf-8");
                                    ByteBuffer byteBuffer = charset.encode(data);
                                    socketChannel.write(byteBuffer,   null,   new   CompletionHandler<Integer,
Void>() {
                                            @Override
                                            public void completed(Integer result, Void attachment) {
                                            }
                                            @Override
                                            public void failed(Throwable exc, Void attachment) {
                                                    try {
                                                            String message = "[클라이언트 통신 안됨: " +
socketChannel.getRemoteAddress() + ": " + Thread.currentThread().getName() + "]";
                                                            Platform.runLater(()->displayText(message));
                                                            connections.remove(Client.this);
                                                            socketChannel.close();
                                                    } catch (IOException e) {}
                                            }
                                    });
                            }
                    }
// 생략
```

(5) UI 생성 코드

ServerExample.java] 채팅 서버

```
// 생략
            ////////////////////////////////////////
            TextArea txtDisplay;
            Button btnStartStop;

            @Override
            public void start(Stage primaryStage) throws Exception {
                    BorderPane root = new BorderPane();
                    root.setPrefSize(500, 300);

                    txtDisplay = new TextArea();
                    txtDisplay.setEditable(false);
```

```
13                      BorderPane.setMargin(txtDisplay, new Insets(0,0,2,0));
14                      root.setCenter(txtDisplay);
15
16                      btnStartStop = new Button("start");
17                      btnStartStop.setPrefHeight(30);
18                      btnStartStop.setMaxWidth(Double.MAX_VALUE);
19                      btnStartStop.setOnAction(e->{
20                              if(btnStartStop.getText().equals("start")) {
21                                      startServer();
22                              } else if(btnStartStop.getText().equals("stop")){
23                                      stopServer();
24                              }
25                      });
26                      root.setBottom(btnStartStop);
27
28                      Scene scene = new Scene(root);
29                      scene.getStylesheets().add(getClass().getResource("app.css").toString());
30                      primaryStage.setScene(scene);
31                      primaryStage.setTitle("Server");
32                      primaryStage.setOnCloseRequest(event->stopServer());
33                      primaryStage.show();
34              }
35
36              void displayText(String text) {
37                      txtDisplay.appendText(text + "\n");
38              }
39
40              public static void main(String[] args) {
41                      launch(args);
42              }
43      //생략
```

~~19.8.7. 채팅 클라이언트 구현~~

19.9 UDP 채널

19.9.1. 발신자 만들기

- DatagramChannel 생성 - Open() 사용
- 데이터 보내기 - send() 사용
- 닫기 - close() 사용

19.9.2. 수신자 만들기

- DatagramChannel 생성 및 포트 바인딩 - open(), bind()
- 데이터 받기
  - 데이터 받기 전까지 receive() 메소드 블로킹, 데이터 받으면 리턴
  - 작업 스레드를 생성해 receive() 메소드 반복적 호출
  - 작업 스레드 종료 방법
    - 작업 스레드의 interrupt() 호출시켜 ClosedByInterruptException 예외 발생
    - DatagramChannel의 close() 호출시켜 AsynchronousCloseException 예외 발생
    - 예외가 발생되면 예외 처리 코드에서 작업 스레드 종료

- 닫기

19.9.3. 수신자와 발신자 실행

- 실행 순서는 상관없지만, 수신자를 먼저 실행하고 발신자를 실행해야만 발신자가 보낸 데이터를 수신자가 모두 받을 수 있다.