NIO란? new IO

NIO의 특징에는 5가지가 있다.

1. 채널방식 2. 버퍼 사용 3. 비동기 지원 4. 블로킹, 넌 플로킹 방식 5. 다양한 파일의 속성정보 제공

이러한 특징이 있는데, 각각에 대해 설명하고 이에 따른 클래스, 메소드가 무엇이 있는지, 어떻게 활용되는지!

1. 채널

스트림은 입, 출력 스트림이 각각 있는 반면, 채널은 "양방향 입출력이 가능"하다.
따로 두 개의 채널을 둘 필요가 없다.

2. 버퍼 사용

NIO에서는 기본적으로 사용하는 것이 *버퍼 방식*임.
버퍼? 하나 하나 바이트를 따로 읽는 것이 아니라, 복수 개의 바이트를 한꺼번에 입력 받아 출력.
(IO에서 BufferedInputStream, BufferedOutputStream을 쓰는 것을 생각하면 된다.)

- **버퍼 사용의 특징**
  1) 성능이 좋아진다. 2) 버퍼 내에서 데이터 위치를 이동해가면서 필요한 부분만 사용할 수 있다.

- **버퍼 클래스들은 "Buffer" 추상 클래스를 모두 상속한다.**

- **바이트 순서: JVM은 big endian으로 동작**

- **버퍼 종류 및 생성**

1) 다이렉트
: OS의 메모리, native C함수를 호출해야 해서 버퍼 생성이 느림.
한 번 생성 해놓고 재사용 - 생성이 느린 것이지, 생성 후에는 더 빠르다!

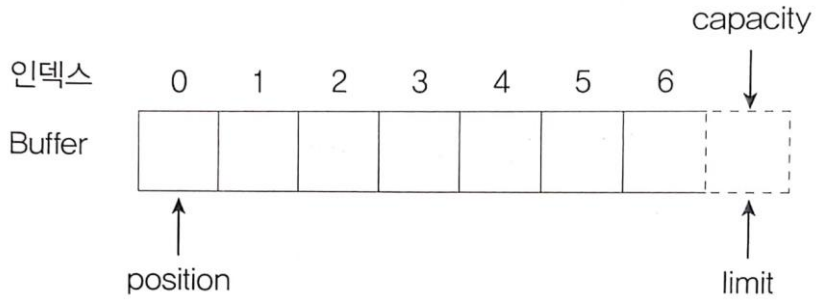
```
ByteBuffer db=ByteBuffer.allocateDirect(200*1024*1024); // 버퍼 생성
```

2) 넌다이렉트
: JVM의 힙 메모리, 버퍼 생성시간이 빠르다. 힙을 사용하여 버퍼 크기가 제한된다.
입출력 시에 임시 버퍼를 거친다, 입출력 성능이 낮은 편

```
ByteBuffer nb=ByteBuffer.allocate(200*1024*1024); // 버퍼 생성

char[] charArray = new Char[100];
CharBuffer cb=CharBuffer.wrap(charArray, 0, 50); // 0부터 50까지만 char형 버퍼 생성
CharBuffer cb1=CharBuffer.wrap("hello world");
```

- **버퍼의 사용; 위치 속성**

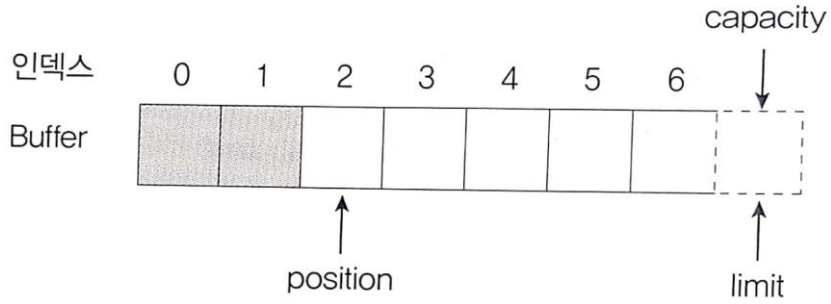
버퍼를 사용하기 위해서는 버퍼의 위치 속성 개념을 잘 알아야 한다!



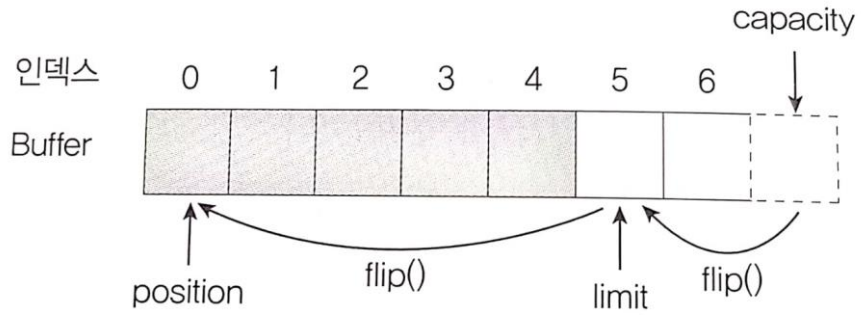
Position: 현재 위치

Capacity: 버퍼 최대 개수

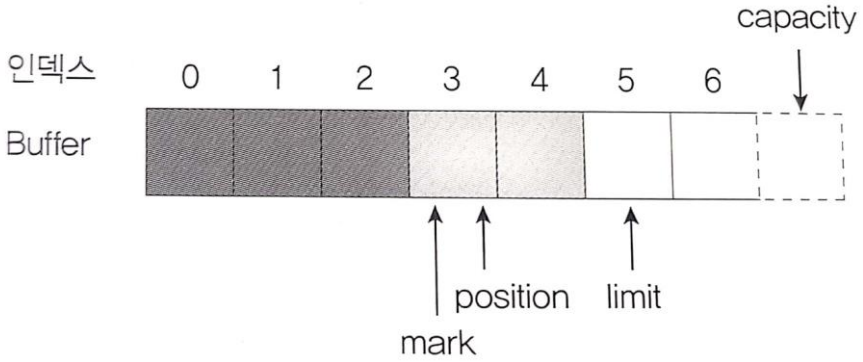
Limit: 버퍼IO의 한계점



버퍼 저장 - 2byte저장



버퍼 읽기 – flip(): limit을 position으로, position을 0으로. 읽을 준비!



3byte읽기 후, 다음 읽기를 위해 mark: 현재 위치 기억

읽은 후 다시 mark로 이동하기	reset()
동일한 데이터를 한번 더 읽기 위해 position을 0으로	rewind()
모든 속성 초기화	clear()
읽지 않은 데이터 뒤에 새로운 데이터 저장	compact()

- 버퍼 메소드: 저장-put(), 읽기-get()

3. 비동기 지원 & 4. 블로킹, 넌 블로킹 방식

블로킹은 하나의 일을 할 때 다른 일들이 대기 상태에 있는 것. IO가 이 방식을 사용함
NIO는 블로킹과 넌 블로킹 둘 다 가능. 여기서 NIO의 블로킹은, interrupt를 사용할 수 있다. 비동기를 지원하기 때문이다.
넌 블로킹은 IO준비 완료된 채널만 선택되기 때문에, 아예 스레드가 블로킹되지 않음.
이 때, 준비 완료된 채널만 선택해주는 것이 Selector이다.

NIO는 대용량이 아닐 때 쓰는 것이 좋다. 무조건 버퍼가 필요하기 때문이다.

5. 파일

다양한 속성 정보를 제공해준다.

1) 파일의 경로:

```
Path path=Paths.get("C:/Temp/dir/file.txt"); // 절대 경로
Path path=Paths.get("./dir/file.txt") // 상대 경로 둘 다 가능!
```

2) 파일의 시스템 정보:

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

3) 파일 생성, 파일의 속성:

```
Path path1=Paths.get("C:/dir");
Path path2=Paths.get("C:/dir/text.txt");
Files.createDirectories(path1);//디렉토리 생성
Files.createFile(path2);//파일 생성
```

4) WatchService: 파일의 내용변화를 감시

```
WatchService ws = FileSystems.getDefault().newWatchService();//WatchService생성
path.register(ws.StandardWatchEventKinds.ENTRY_CREATE)//생성을 감시하는 ws 등록
```

```
while(true) {
	WatchKey watchKey = watchService.take();
	List<WatchEvent<?>> list = watchKey.pollEvents();
	for(WatchEvent watchEvent : list) {//이벤트 종류 얻기
		Kind kind = watchEvent.kind();//감지된 Path 얻기
		Path path = (Path)watchEvent.context();
		if(kind == StandardWatchEventKinds.ENTRY_CREATE) {//생성되었을 경우
			Platform.runLater(()->textArea.appendText(path.getFileName() + "₩n"));
		} else if(kind == StandardWatchEventKinds.ENTRY_DELETE) {//삭제되었을 경우
			Platform.runLater(()->textArea.appendText(path.getFileName() + "₩n"));
		} else if(kind == StandardWatchEventKinds.ENTRY_MODIFY) {//변경
			Platform.runLater(()->textArea.appendText(path.getFileName() + "₩n"));
		} else if(kind == StandardWatchEventKinds.OVERFLOW) {}
}
```

5-1. 파일 처리

파일을 처리하기 위해서는 “FileChannel”을 이용한다.

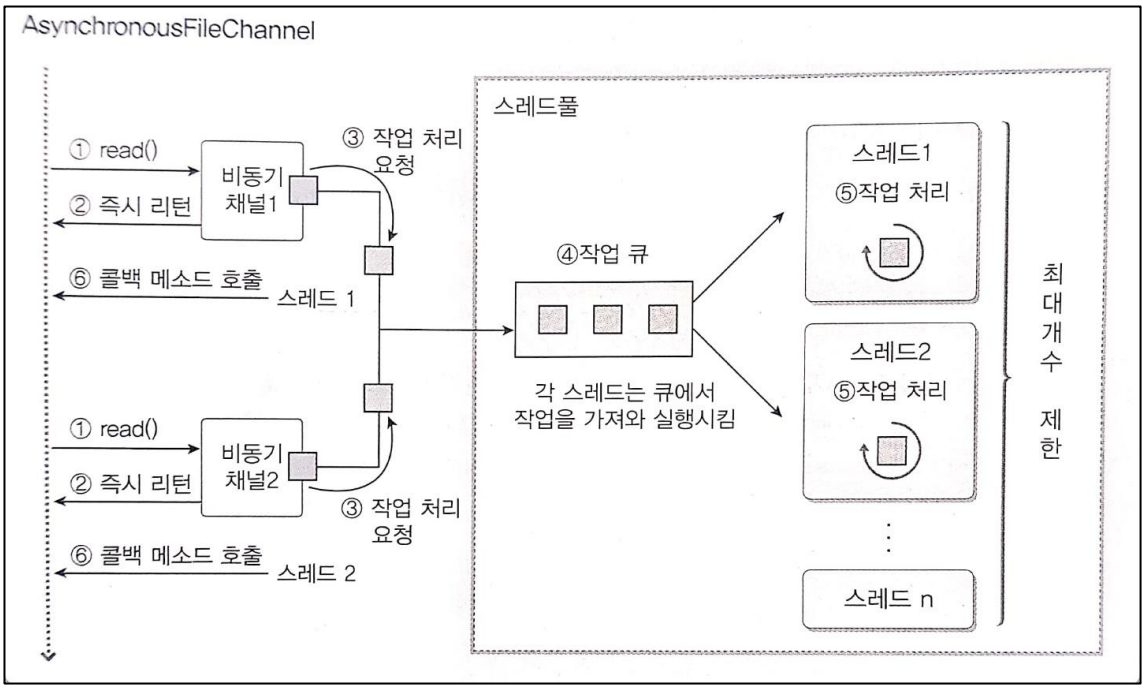
- 파일 read, write

```
/* 읽고 쓸 수 있는 파일을 생성*/
FileChannel fileChannel = FileChannel.open(
        Paths.get("C:Temp/dir/text.txt"),
        StandardOpenOption.CREATE_NEW,
        StandardOpenOption.READ
        StandardOpenOption.WRITE);

/* String을 Buffer로 변환 저장*/
String data = "안녕하세요";
Charset charset = Charset.defaultCharset();//charset을 지정해주어야 한다.
ByteBuffer byteBuffer = charset.encode(data);

/* 해당 내용을 파일에 저장*/
//write를 read로 바꾸면 파일 읽기.
int byteCount = fileChannel.write(byteBuffer);//return : 쓰여진 바이트 수
System.out.println("file.txt : " + byteCount + " bytes written");

fileChannel.close();//파일 닫기
```

- 파일 복사
  1) FileChannel 2개를 생성하여 복사 2) NIO의 Files클래스 copy() 메소드 사용

- **파일 비동기**

: 대용량 파일의 입출력 작업을 위해 비동기 파일!!!




```
public class AsynchronousFileChannelReadExample {
	public static void main(String[] args) throws Exception {
		//스레드풀 생성
		ExecutorService executorService = Executors.newFixedThreadPool(
			Runtime.getRuntime().availableProcessors()
		);

		for(int i=0; i<10; i++) {
			Path path = Paths.get("C:/Temp/file" + i + ".txt");

			//비동기 파일 채널 생성
			AsynchronousFileChannel fileChannel = AsynchronousFileChannel.open(
				path,
				EnumSet.of(StandardOpenOption.READ),
				executorService
			);

			ByteBuffer byteBuffer = ByteBuffer.allocate((int)fileChannel.size());

			//첨부 객체 생성
			class Attachment {
				Path path;
				AsynchronousFileChannel fileChannel;
				ByteBuffer byteBuffer;
			}
			Attachment attachment = new Attachment();
			attachment.path = path;
```

```
                        attachment.fileChannel = fileChannel;
                        attachment.byteBuffer = byteBuffer;

                        //CompletionHandler 객체 생성
                        CompletionHandler<Integer, Attachment> completionHandlernew =
                                new CompletionHandler<Integer, Attachment>() {
                                @Override
                                public void completed(Integer result, Attachment attachment) {
                                        attachment.byteBuffer.flip();

                                        Charset charset = Charset.defaultCharset();
                                        String                          data                            =
charset.decode(attachment.byteBuffer).toString();

                                        System.out.println(attachment.path.getFileName()  +  " : " +
data + " : " + Thread.currentThread().getName());
                                        try { fileChannel.close(); } catch (IOException e) {
                                                //e.printStackTrace();
                                        }
                                }
                                @Override
                                public void failed(Throwable exc, Attachment attachment) {
                                        exc.printStackTrace();
                                        try { fileChannel.close(); } catch (IOException e) {}
                                }
                        };

                        //파일 읽기
                        fileChannel.read(byteBuffer, 0, attachment, completionHandlernew);
                }

                //스레드풀 종료
                //Thread.sleep(1000);
                executorService.shutdown();
        }
}
```