Chapter 15. 셀렉터

Originally made by Prof. Hanku Lee
Modified by Mingyu Lim

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

1. 셀렉터 개요

❑ Reactor 패턴

클라이언트들의 요청을 앞단의 큐에 저장 -> 큐를 모니터링 하는 스레드에 이벤트 전달 -> 모니터링 하는 스레드가 큐에 저장된 요청 방향 분석 -> 적절한 프로세스 로직으로 보내주어 해당 요청 처리

: NIO 비블록킹 서버 구현의 밑바탕 (Reactor 패턴)

: Selector 는 바로 Reactor 의 역할을 한다.

❑ 셀렉터 역할

SelectableChannel을 자신에게 등록 -> 등록된 SelectableChannel 의 이벤트 요청 구분 -> 적절한 서비스 제공자에게 보내 처리

: 즉 멀티플렉스 IO 가능하게 해준다.

2. 기존의 네트워크 프로그래밍 모델

❑ 블록킹 IO

- 블록킹 IO 란

: 특정 디바이스에서 읽기, 쓰기 작업을 할때 데이터를 이용할수 있을 때까지 해당 IO 작업을 수행하려던 스레드가 아무것도 하지 않고 대기하는 것

- 블록킹 IO 예

: 소켓에서 BufferedRead 객체를 얻어 이 스트림으로 부터 데이터를 읽어 오기 위해 readLine() 메소드 호출

: readLine() 메소드 호출했을때 해당 스트림으로부터 개행문자(\n)를 읽을 때까지 이 스레드는 블록킹

2. 기존의 네트워크 프로그래밍 모델(계속)

❑ 블록킹 IO

```
Socket s = serverSocket.accept()
                ↓
InputStream in = s.getInputStream()
OutputStream out = s.getOutputStream()

                ↓
            in.read()
                ↓
            out.write()
                ↓
            s.close()
```

: 표시된 부분들이 블록킹 되는 부분

: 위의 구조의 서버는 동시에 단 한명의 요청밖에 수행 못함
(싱글 스레드 모델)

2. 기존의 네트워크 프로그래밍 모델(계속)

❑ 블록킹 IO

- 싱글 스레드 모델 단점 보완 방법(멀티스레드 모델 방법)

```
ServerSocket ss = new ServerSocket(4567);
while (true)   {
            socket s = ss.accept();
            …
            // 동시에 여러 클라이언트들의 요청을 수행하기 위해 별도의 스레드를
               만들어서 처리한다.
            Service service = new Service(s);
            service.start();
```

: 스레드 개수가 접속된 클라이언트 수만큼 증가하는 문제점 내포

: 클라이언트당 하나의 Service 스레드 만들어야 함

: 표시된 부분이 Service 클래스가 스레드를 상속하게 만듦

2. 기존의 네트워크 프로그래밍 모델(계속)

❑ 멀티 스레드 모델의 문제점

- 많은 스레드 생성에 따른 스레드 컨텍스트 스위치 부하
  : 보통 경량 서버의 경우 천여개 정도의 스레드가 생성되면 스레드 컨텍스트 부하로 급격한 성능 저하 발생

- 스레드 자체가 CPU와 고유 스택을 갖는 데 따른 컴퓨터 리소스 부하

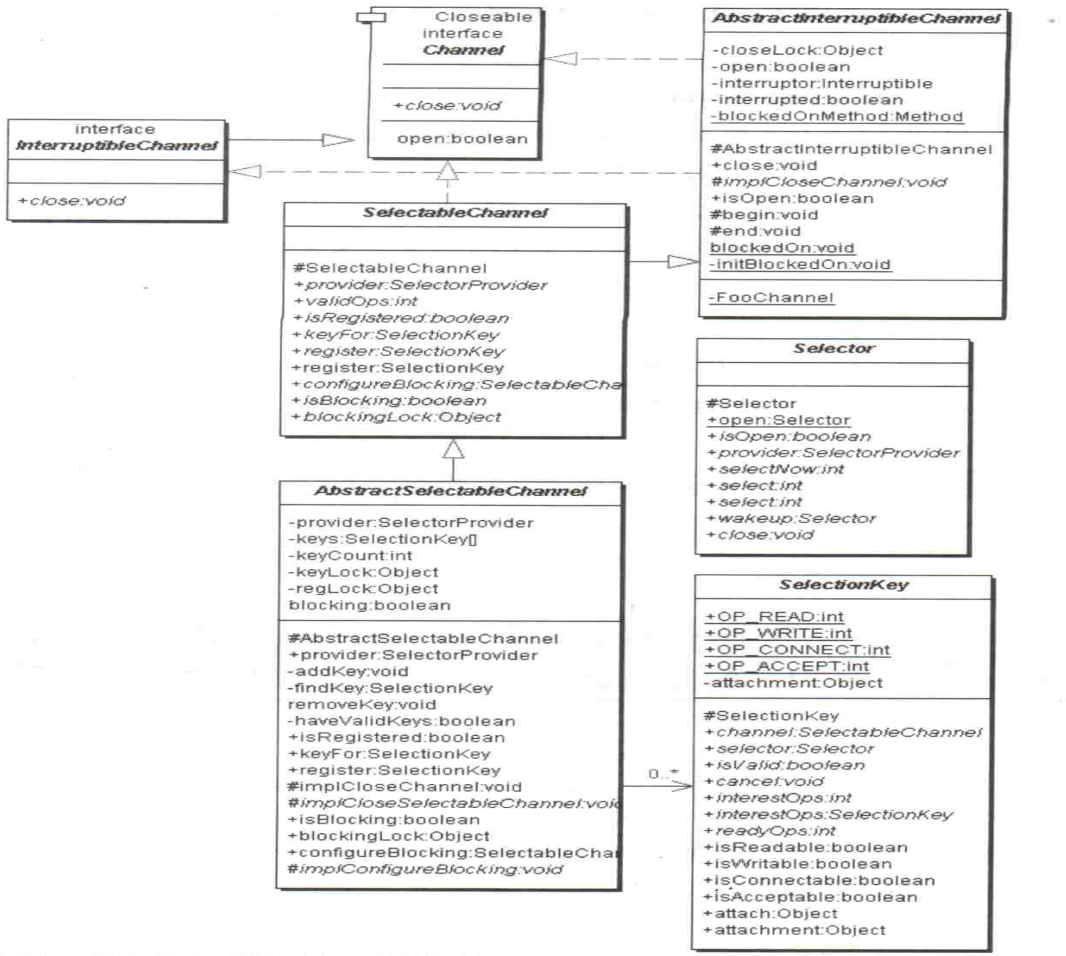
- 클라이언트의 빈번한 접속과 종료에 따라 많은 가비지가 생성
  : 클라이언트와 데이터를 주고받는 과정에서 JVM 힙 영역으로 데이터의 복사가 빈번하게 발생하므로 많은 가비지 발생
  : 스레드의 잦은 생성과 소멸에 따른 가비지도 발생

- 클라이언트가 접속할 때 마다 매번 스레드를 새로 생성하는 부담

- 서버의 메모리가 부족해서 OutofMemoryException이 발생할 수 있는 가능성

3. 비블록킹 모델

❑ Select 관련 클래스의 클래스 다이어그램



[그림 15-2] Select 관련 클래스들의 클래스 다이어그램

3. 비블록킹 모델(계속)

❑ Workflow(비록킹 사용해서 멀티스레드 서버와 같은 동작)

- 채널을 Selector에 등록하면 등록에 관련된 채널과 Selector 사이의 관계를 캡슐화한 selectionKey 가 selector에 저장

- 어떤 채널이 자신이 등록한 모드에 대해 동작할 준비가 되면 SelectionKey는 그 준비상태를 내부적으로 저장

- Selector가 select() 메소드를 호출해서 자신에게 등록된 모든 SelectionKey 들을 검사하는데, 바로 동작할 준비가 되어 있는지 아닌지를 검사

멀티플렉스 모델의 서버를 만들기 위해 핵심적인 역할을 하는 것은 Selector, SelectableChannel, SelectionKey 이다.

4. SelectableChannel

❑ SelectableChannel API

```
public abstract class SelectableChannel extends AbstractChannel implements
    Channel {

        // 블록킹 – 비블록킹 모드 설정에 관련된 메소드들
        public abstract void configureBlocking (boolean block)
        throws IOException
        public abstract boolean isBlocking();
        public abstract Object blockingLock();

        // 채널을 Selector에 등록하는 것과 관련된 메소드들
        public abstract SelectionKey register(Selector sel, int ops)
        throws ClosedChannelException;
        public abstract SelectionKey register (Selector sel, int ops, Object att)
        throws ClosedchannelException;
        public abstract boolean isRegistered();
        public abstract SelectionKey keyFor(Selector sel);
        public abstract int validOps();
```

4. SelectableChannel (계속)

❑ 채널을 Selector에 등록하기

- 등록할 채널을 우선 비블록킹 모드로 설정
- SelectableChannel의 register() 메소드를 호출해서 Selector 에 등록
  (블록킹 모드로 된 채널을 등록시 IllegalBlockingModeException 발생)

모든 채널은 생성시 기본적으로 블록킹 모드로 설정되므로 반드시
`configureBlocking (false)` 호출해서 비블록킹 모드로 만들어야 한다.

- 닫힌 (close) 채널을 register () 메소드로 등록 하려면 예외발생
  (ClosedChannelException 던지도록 만들어 놓음)

```
public abstract SelectionKey register (Selector sel, int ops) throws ClosedChannelException

public abstract SelectionKey register (Selector sel, int ops, Object att) throws
ClosedChannelException
```

4. SelectableChannel (계속)

❑ 채널을 Selector에 등록하기

- Selector 에 등록할수 있는 이벤트

```
public abstract class SelectionKey {
        public static final int OP_READ = 1 << 0 ;
        public static final int OP_WRITE = 1 << 2 ;
        public static final int OP_CONNECT = 1 << 3 ;
        public static final int OP_ACCEPT = 1 << 4 ;
```

- 서버 소켓채널을 Selector 에 등록하는 템플릿

```
ServerSocketChannel server = ServerSocketChannel.open();
server.configureBlocking(false);

ServerSocket socket = server.socket();
SocketAddress addr = new InetSocketAddress(port);
socket.bind(addr);

server.register(selector, SelectionKey.OP_ACCEPT);
```

4. SelectableChannel (계속)

❑ 채널을 Selector에 등록하기

- 소켓 채널을 Selector에 등록하는 템플릿

```
…
ServerSocketChannel server = (ServerSocketChannel) key.channel();
SocketChannel sc = server.accept();
boolean isRegist = registerChannel(selector, sc, SelectionKey.OP_READ);
…

private boolean registerChannel (Selector selector, SelectableChannel sc, int ops)
            throws ClosedChannelException, IOException {
                        if (sc == null) {
                           return false;
                        }
                        sc,configureBlocking(false);
                        sc.register(selector, ops);

                        return true;
            }
```

4. SelectableChannel (계속)

❑ 채널을 Selector에 등록하기

- Selector에 등록 가능한 채널별 모드

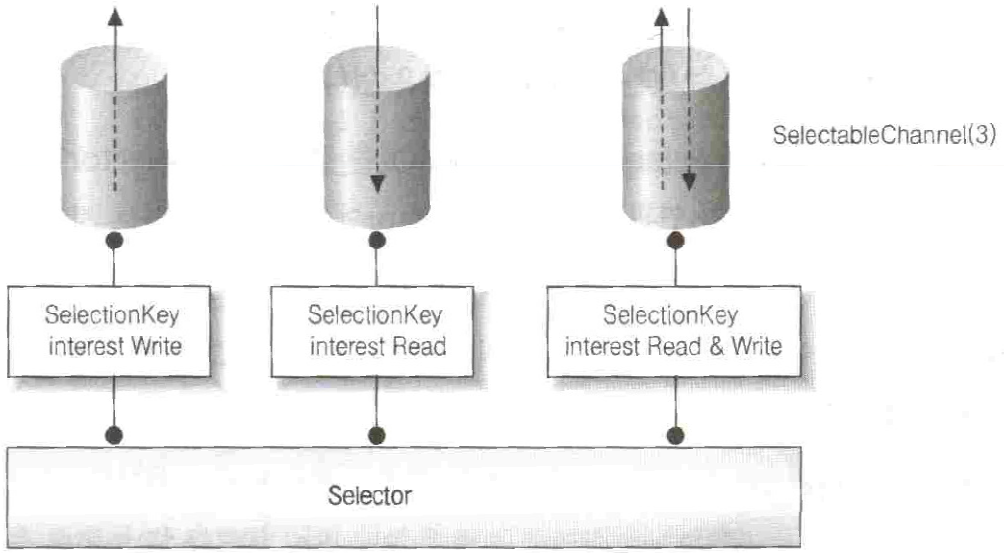
구분	내용
ServerSocketChannel	OP_ACCEPT
SocketChannel	OP_CONNECT, OP_READ, OP_WRITE
DatagramChannel	OP_READ, OP_WRITE
Pipe.sourceChannel	OP_READ
Pipe.SinkChannel	OP_WRITE

: 이벤트 한 개만을 등록할수 있는것이 아니라 여러개도 등록가능

4. SelectableChannel (계속)

❑ 채널을 Selector에 등록하기

- 채널의 이벤트 등록에 다른 Selector 의 반응



[그림 15-3] 채널의 이벤트 등록에 따른 Selector의 반응

4. SelectableChannel (계속)

❑ 채널을 Selector에 등록하기

- 남은 API 메소드

```
public abstract boolean isRegistered();
public abstract SelectionKey keyFor(Selector sel);
public abstract int validOps();
```

: isRegistered()는 채널이 Selector에 등록되어 있는지를 질의하는 메소드
(true 등록 , false 미등록)
-> true 가 리턴되더라도 채널이 Selector가 이용할수 있는 상태라는 보장은 없다.

: KeyFor() 는 파라미터로 넘어온 Selector에 대해 이 채널이 등록한 상태라면 등록에 중계자 역할을 하는 SelectionKey를 리턴 / 미등록하였다면 null 을 리턴

: validOps() 채널이 Selector에 등록한 이벤트 값을 리턴

5. SelectionKey

❑ SelectionKey API

```
public abstract class SelectionKey {
        public static final int OP_READ
        public static final int OP_WRITE
        public static final int OP_CONNECT
        public static final int OP_ACCEPT

        public abstract SelectableChannel channel ();
        public abstract Selector selector  ();

        public abstract void cancel();
        public boolean isValid();

        public abstract int interestOps();
        public abstract void interestOps (int ops);
        public abstract int readyOps();

        public final boolean isReadable();
        public final boolean isWritable();
        public final boolean isConnectable();
        public final boolean isAcceptable();

        public final Object attach (Object ob);
        public final Object attachment ();
    }
```

5. SelectionKey(계속)

❑ SelectionKey에 대해서

- 특정 채널과 selector 사이의 등록 관계를 캡슐화
  (특정 Selector에 register() 메소드로 등록하면 연관관계를 표현하는 SelectionKey 객체가 생성)

- 생성된 객체는 두 가지의 핵심적인 역할을 하는 Set을 가짐
  1. interest set : SelectableChannel이 register()로 Selector에 등록한 오퍼레이션들(ops) 을 저장한다.
  2. ready set : SelectableChannel에서 이벤트가 발생하면 그 이벤트를 저장한다.

5. SelectionKey(계속)

❑ SelectionKey의 오퍼레이션별 발생 이벤트

구분	내용
OP_ACCEPT	클라이언트가 ServerSocketChannel에 접속을 시도했을 때 발생
OP_CONNECT	서버가 클라이언트의 접속을 허락했을때 발생
OP_READ	서버가 클라이언트의 read 할수 있을때 발생
OP_WRITE	서버가 클라이언트에게 응답을 write 할수 있을때 발생

5. SelectionKey(계속)

❑ SelectionKey의 메소드

```
public abstract SelectableChannel channel ();
public abstract Selector selector  ();
```

: channel() -> SelectionKey 자신과 연관된 채널을 리턴

: selector() -> 자신관 연관된 Selector을 리턴

```
public abstract void cancel();
public boolean isValid();
```

: cancel() -> SelectionKey를 해당 Selector에서 삭제 즉 등록을 해제하기 위한 메소드

: isValid() -> SelectionKey가 유효한지 아닌지를 질의하는 메소드
SelectionKey가 cancel() 메소드 호출로 삭제, 채널이 닫히든지 또는 Selector 가 닫히면 유효한 키가 아니라는 의미의 false 리턴 / 정상적인 경우에는 true가 리턴

5. SelectionKey(계속)

❑ SelectionKey의 메소드

```
public abstract int interestOps();
public abstract void interestOps (int ops);
public abstract int readyOps();
```

: interestOps() -> 채널이 등록한 이벤트 값의 집합인 "interest set" 값 리턴

: interestOps(int ops) -> "interest set"을 새롭게 설정하는 메소드

: readyOps() -> 채널이 등록한 이벤트중 이벤트에 대한 값을 리턴

: readyOps() 사용 템플릿

```
if ((key.readyOps() & SelectionKey.OP_READ) !=0)  {
      …
      // 읽기 작업을 수행한다.
      …
       }
```

-> 준비된 작업이 '읽기' 인지 확인후 읽기 작업인 경우 if문 수행

5. SelectionKey(계속)

❑ SelectionKey의 메소드

```
public final boolean isReadable();
public final boolean isWritable();
public final boolean isConnectable();
public final boolean isAcceptable();
```

: SelectionKey에 수행 준비된 이벤트가 읽기, 쓰기, 연결, 접속 인지를 질의 하는 메소드

: 사용예(같은 의미)

```
if (key.isWritable()) {
        …
    }
```

=

```
if ((key.readyOps() & SelectionKey.OP_WRITE) !=0 {
        …
    }
```

5. SelectionKey(계속)

❑ SelectionKey의 메소드

```
public final Object attach (Object ob);
public final Object attachment ();
```

: attach (Object ob) -> SelectionKey에 참조할 객체를 추가

: attachment () -> 해당키에 참조할 객체가 있으면 그 객체를 리턴
그렇지 않으면 null을 리턴

: SelectionKey가 삭제되어 가바지 컬렉션 대상이 되더라도 attach()메소드로 첨부된 객체는 삭제되지 않는다. SelectionKey의 삭제와 함께 반드시 첨부된 객체를 같이 제거해야 한다.

6. Selector

❑ Selector API

```
public abstract class Selector {
        public static Selector open() throws IOException ;
        public abstract boolean isOpen() ;
        public abstract void close() throws IOException;

        public abstract Set keys();
        public abstract Set selectedKeys();

        public abstract int select() throws IOException;
        public abstract int select(long timeout) throws IOException;
        public abstract int selectNow() throws IOException;
        public abstract void wakeup();
  }
```

- Seletor 는 등록된 채널들이 발생시킨 이벤트에 대해 적절한 처리 핸들러로 그 요청을 분기시켜주는 컨드롤러의 역할

6. Selector(계속)

❑ Selector API

```
Selector selector = Selector.open();
```

: 버퍼나 채널과 마찬가지로 생성자를 사용해서 객체를 만들 수 없고
open() 팩토리 메소드만을 사용해서 생성

: isOpen() 메소드 -> open() 통해 Selector를 생성하고 사용 가능한 상태
인지를 질의하는 메소드

6. Selector(계속)

❑ Selector API

```
public abstract Set keys();
public abstract Set selectedKeys();
```

: Selector는 다음과 같이 내부적으로 세 가지 Key Set을 관리

1. 등록된 키 집합 (Registered Key Set)
   - Selector에 현재 등록된 모든 SelectionKey의 집합
   - 집합에 있는 SelectionKey가 항상 유효한 것은 아님
   - 집합은 keys() 통해 얻을수 있다.

2. 선택된 키 집합 (Selected Key Set)
   - '등록된 키 집합' 의 부분 집합
   - "ready set" 비어있지 않은 SelectionKey 들이 Selector의 selection 메소드의 호출을 통해 '선택된 키 집합' 에 추가
   - selectedKeys()를 통해 '선택된 키 집합' 을 얻을 수 있다.

6. Selector(계속)

❑ Selector API

: Selector는 다음과 같이 내부적으로 세 가지 Key Set을 관리

3. 취소된 키 집합 (Cancelled Key Set)
   - 채널이 Selector에 등록되었을 경우, 등록 해제하고 싶으면 cancel() 사용
   - 취소된 SelectionKey 는 곧바로 유효하지 않은키로 설정되고 취소된 키 집합에 추가
   - SelectionKey가 Selector에서 등록이 곧바로 해제되지 않음

6. Selector(계속)

❑ Selector API

- selection 메소드의 동작 워크플로우

1. ‘취소된 키 집합’ 체크 / 집합이 비어있지 않다면 집합에 저장된 각각의 키들은 Selector가 관리하는 세가지 키 집합에서 모두 삭제되어 각 키와 연관된 채널이 Selector에서 등록이 해제 / 작업이 완료되면 이 집합은 빈상태로 변경

2. ‘등록된 키 집합’에 저장된 각 SelectionKey를 확인 / ready set이 비어 있지 않은, 즉 이벤트가 발생한 SelectionKey들을 ‘선택된 키 집합’에 넣는다. 만약 ‘선택된 키 집합’에 해당 SelectionKey가 존재한다면 단순히 SelectionKey를 업데이트만 한다. / 존재하지 않으면 새로 ‘선택된 키 집합’ 에 추가된다.

3. 애플리케이션이 Selector의 selectedKeys() 메소드를 호출해서 Selector에 저장된 ‘선택된 키 집합’을 가져오고 그 안에 저장된 각각의 SelectionKey들의 이벤트 형식(accept, connect, read, write) 에 따라 적절한 핸들러에게 처리를 넘긴다.

6. Selector(계속)

❑ Selector API

```
int n = selector.select();
```

: select() 메소드는 블록킹 되는 메소드
: select() 호출한 시점에 '선택된 키 집합'이 비어 있다면 이메소드는 해당 집합에 이용 가능한 SelectionKey가 추가 될때까지 블록킹
: 이용 가능한 SelectionKey가 추가되면 이 메소드는 바로 1,2,3 워크플로우를 실행


```
int n = selector.select(1000);
```

: 파라미터로 주어진 밀리 세컨드마다 앞서의 select() 메소드와 동일한 동작

6. Selector(계속)

❑ Selector API

```
int n = selector.selectNow();
```

: selectNow() 는 비블록킹
: 이용 가능한 채널이 없을 경우에는 0을 리턴
: 이용 가능한 채널이 있다면 '선택된 키 집합' 안에 들어있는
SelectionKey들의 개수를 리턴

: 다음과 같은 의미

-> int n = selector.select(0)

6. Selector(계속)

❑ Selector API

: wakeup() -> selection 메소드들 중 하나를 수행하던중 스레드가 블록킹 되어 있을 경우 이렇게 블록킹된 스레드를 깨우는데 사용

7. 비블록킹 서버 만들기

❑15-1 SimpleChatServer.java

```
public void initServer() {
try {
  selector = Selector.open();
  serverSocketChannel = ServerSocketChannel.open();
  serverSocketChannel.configureBlocking(false);
  serverSocket = serverSocketChannel.socket();
  InetSocketAddress isa = new InetSocketAddress(HOST, PORT);
  serverSocket.bind(isa);

  serverSocketChannel.register(selector, SelectionKey.OP_ACCEPT);
} catch (IOException e) {
  log(Level.WARNING, "SimpleChatServer.initServer()", e);
  }
}
```

- Open()으로 셀렉터와 서버소켓채널 생성
- 비블록킹 모드 설정후, register()로 서버소켓채널을 셀렉터에 등록

7. 비블록킹 서버 만들기

❑15-1 SimpleChatServer.java

```
public void startServer() {
info("Server is started..");
try {
  while (true) {
    info("요청을 기다리는 중..");
    selector.select();
    Iterator it = selector.selectedKeys().iterator();
    while (it.hasNext()) {
      SelectionKey key = (SelectionKey) it.next();
      if (key.isAcceptable()) {
        accept(key);
      } else if (key.isReadable()) {
        read(key);
      }
      it.remove();
    }
  }
} catch (Exception e) {
  log(Level.WARNING, "SimpleChatServer.startServer()", e);
}
}
```

- Selector.selectedKeys()로 준비된 요청을 저장하는 키 set 반환
- 처리한 키는 해당 set에서 반드시 삭제

7. 비블록킹 서버 만들기

❑15-2 SimpleChatClient.java

```
public void initServer() {
try {
  selector = Selector.open();
  sc = SocketChannel.open(new InetSocketAddress(HOST, PORT));
  sc.configureBlocking(false);
  sc.register(selector, SelectionKey.OP_READ);
} catch (IOException e) {
  log(Level.WARNING, "SimpleChatClient.initServer()", e);
}
}
```

```
public void startServer() {
  startWriter();
  startReader();
}

private void startWriter() {
  info("Writer is started..");
  Thread t = new MyThread(sc);
  t.start();
}
```

7. 비블록킹 서버 만들기

❑15-2 SimpleChatClient.java

```
class MyThread extends Thread {
  private SocketChannel sc = null;
  public MyThread(SocketChannel sc) {
    this.sc = sc;
  }
  public void run() {
    ByteBuffer buffer = ByteBuffer.allocateDirect(1024);
    try {
      while (!Thread.currentThread().isInterrupted()) {
        buffer.clear();
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        String message = in.readLine();
        if (message.equals("quit") || message.equals("shutdown")) {
          System.exit(0);
        }
        buffer.put(message.getBytes());
        buffer.flip();
        sc.write(buffer);
      }
    } catch (Exception e) {
      log(Level.WARNING, "MyThread.run()", e);
    } finally {
      clearBuffer(buffer);
    }
  }
}
```

- SimpleChatClient의 inner 클래스
- 사용자가 입력한 데이터를 소켓채널로 전송

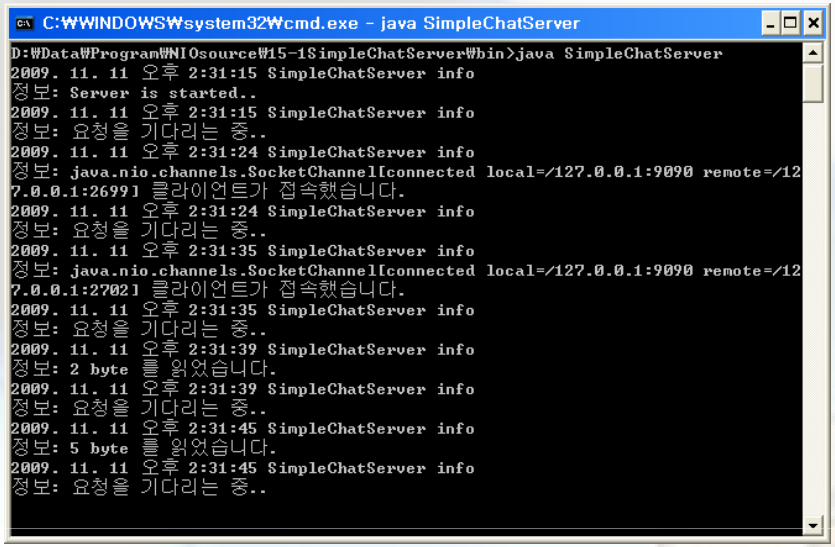
7. 비블록킹 서버 만들기

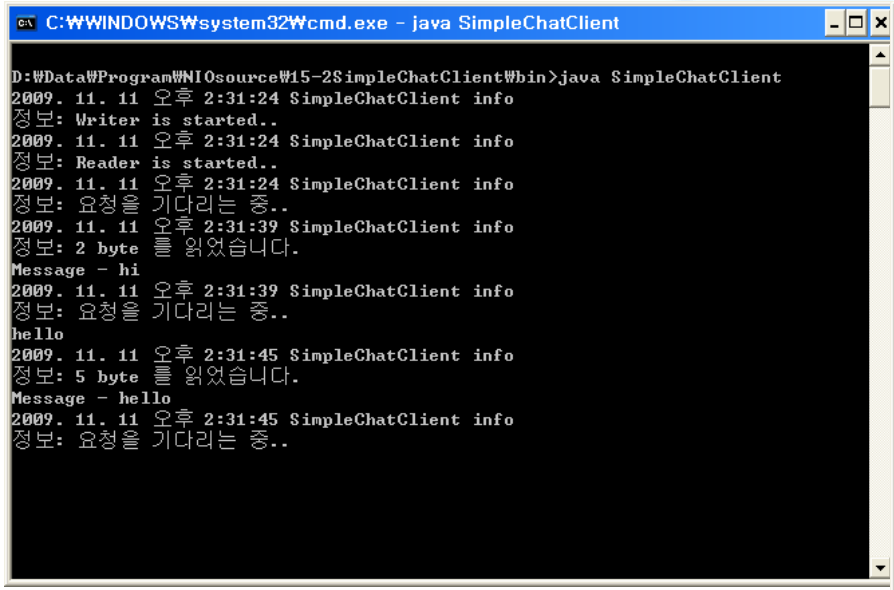
❑15-2 SimpleChatClient.java

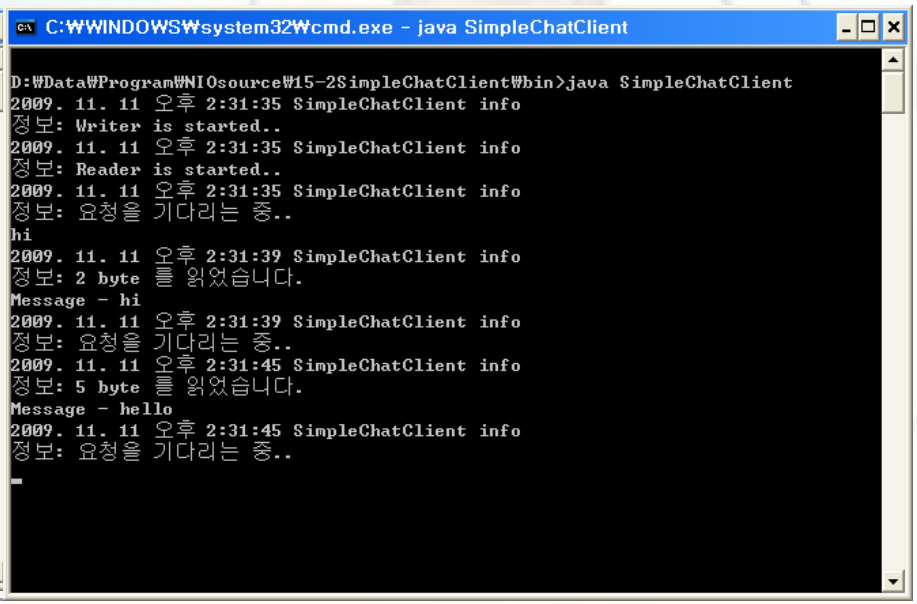
```
private void startReader() {
info("Reader is started..");
try {
  while (true) {
    info("요청을 기다리는 중..");
    selector.select();
    Iterator it = selector.selectedKeys().iterator();
    while (it.hasNext()) {
      SelectionKey key = (SelectionKey) it.next();
      if (key.isReadable()) {
        read(key);
      }
      it.remove();
    }
  }
} catch (Exception e) {
  log(Level.WARNING, "SimpleChatClient.startServer()", e);
}
}
```

- 서버의 broadcast()메소드와 유사
- Data = decoder.decode(buffer).toString();
  - ◆ Charset이 "EUC-KR" 인 디코더를 이용해서 CharBuffer로 만든후 이를 String으로 변환

7. 비블록킹 서버 만들기


C:₩WINDOWS₩system32₩cmd.exe - java SimpleChatServer
D:₩Data₩Program₩NIOsource₩15-1SimpleChatServer₩bin>java SimpleChatServer
2009. 11. 11 오후 2:31:15 SimpleChatServer info
정보: Server is started..
2009. 11. 11 오후 2:31:15 SimpleChatServer info
정보: 요청을 기다리는 중..
2009. 11. 11 오후 2:31:24 SimpleChatServer info
정보: java.nio.channels.SocketChannel[connected local=/127.0.0.1:9090 remote=/127.0.0.1:2699] 클라이언트가 접속했습니다.
2009. 11. 11 오후 2:31:24 SimpleChatServer info
정보: 요청을 기다리는 중..
2009. 11. 11 오후 2:31:35 SimpleChatServer info
정보: java.nio.channels.SocketChannel[connected local=/127.0.0.1:9090 remote=/127.0.0.1:2702] 클라이언트가 접속했습니다.
2009. 11. 11 오후 2:31:35 SimpleChatServer info
정보: 요청을 기다리는 중..
2009. 11. 11 오후 2:31:39 SimpleChatServer info
정보: 2 byte 를 읽었습니다.
2009. 11. 11 오후 2:31:39 SimpleChatServer info
정보: 요청을 기다리는 중..
2009. 11. 11 오후 2:31:45 SimpleChatServer info
정보: 5 byte 를 읽었습니다.
2009. 11. 11 오후 2:31:45 SimpleChatServer info
정보: 요청을 기다리는 중..


C:₩WINDOWS₩system32₩cmd.exe - java SimpleChatClient
D:₩Data₩Program₩NIOsource₩15-2SimpleChatClient₩bin>java SimpleChatClient
2009. 11. 11 오후 2:31:24 SimpleChatClient info
정보: Writer is started..
2009. 11. 11 오후 2:31:24 SimpleChatClient info
정보: Reader is started..
2009. 11. 11 오후 2:31:24 SimpleChatClient info
정보: 요청을 기다리는 중..
2009. 11. 11 오후 2:31:39 SimpleChatClient info
정보: 2 byte 를 읽었습니다.
Message - hi
2009. 11. 11 오후 2:31:39 SimpleChatClient info
정보: 요청을 기다리는 중..
hello
2009. 11. 11 오후 2:31:45 SimpleChatClient info
정보: 5 byte 를 읽었습니다.
Message - hello
2009. 11. 11 오후 2:31:45 SimpleChatClient info
정보: 요청을 기다리는 중..


C:₩WINDOWS₩system32₩cmd.exe - java SimpleChatClient
D:₩Data₩Program₩NIOsource₩15-2SimpleChatClient₩bin>java SimpleChatClient
2009. 11. 11 오후 2:31:35 SimpleChatClient info
정보: Writer is started..
2009. 11. 11 오후 2:31:35 SimpleChatClient info
정보: Reader is started..
2009. 11. 11 오후 2:31:35 SimpleChatClient info
정보: 요청을 기다리는 중..
hi
2009. 11. 11 오후 2:31:39 SimpleChatClient info
정보: 2 byte 를 읽었습니다.
Message - hi
2009. 11. 11 오후 2:31:39 SimpleChatClient info
정보: 요청을 기다리는 중..
2009. 11. 11 오후 2:31:45 SimpleChatClient info
정보: 5 byte 를 읽었습니다.
Message - hello
2009. 11. 11 오후 2:31:45 SimpleChatClient info
정보: 요청을 기다리는 중..