1주차: 과목 OT

2주차: C언어로 구현하는 TCP/IP 소켓 프로그래밍

3주차: C언어로 구현하는 TCP/IP 소켓 프로그래밍

4주차: JAVA언어로 구현하는 스레드/소켓 통신 프로그래밍

5주차: JAVA언어로 구현하는 스레드/소켓 통신 프로그래밍

6주차: JAVA언어로 구현하는 스레드/소켓 통신 프로그래밍

7주차: 안드로이드 (프로젝트 만들기)

**1주차**

**통신의 구성요소**

송신자(Sender): 메시지를 보내는 장치 ex. 컴퓨터, 전화기, 비디오 카메라 등

수신자(Receiver): 메시지를 받는 장치 ex. 컴퓨터, 전화기, TV

전송매체(Medium): 송신자에서 수신자까지 메시지를 전달하는 물리적인 경로 ex. 꼬임선, 동축선, 광케이블

프로토콜(Protocol): 데이터 통신 수행 규칙들의 집합(상호 합의)

메시지(Message): 전송되는 정보(데이터) ex. 문자, 소리, 영상, 그림, 이들의 조합

**네트워크 개념**

**인터넷 구성요소**

컴퓨터 노드: 송신자 혹은 수신자의 역할

인터넷: 전령 혹은 매체의 역할

전화망과 동일한 흐름

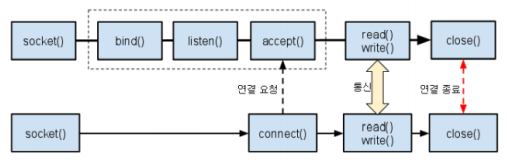
소켓: 전화기의 역할을 함. 인터넷과의 접점 혹은 관문

Node를 인터넷에 연결하기

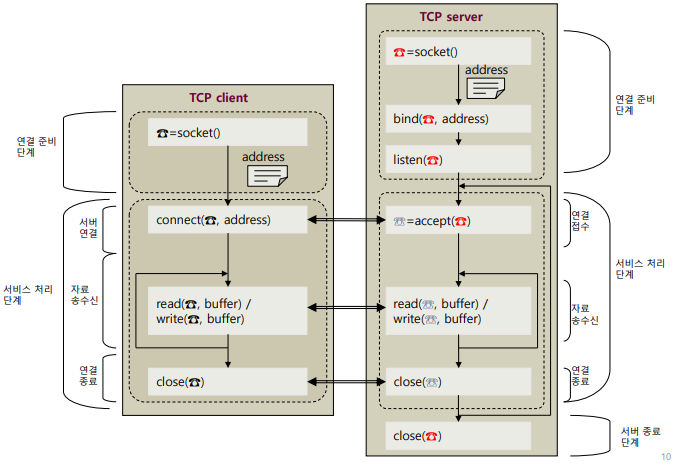
- 소켓에 전체 인터넷 노드에서 유일하게 식별가능한 주소(IP)를 부여한다

- 프로그램을 찾을 수 있도록 포트 번호를 bind 한다

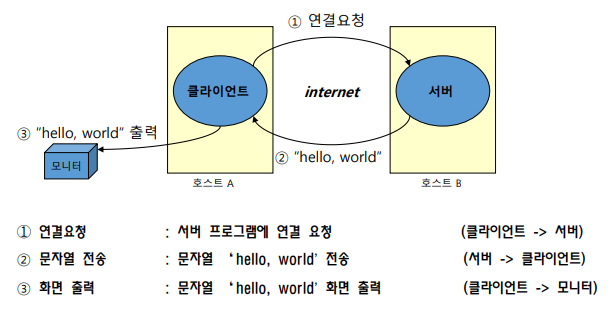
서버와 클라이언트 네트워크 프로그램 흐름



**TCP 소켓 프로그래밍**



**Hello, world 출력 소켓 프로그램**



정리

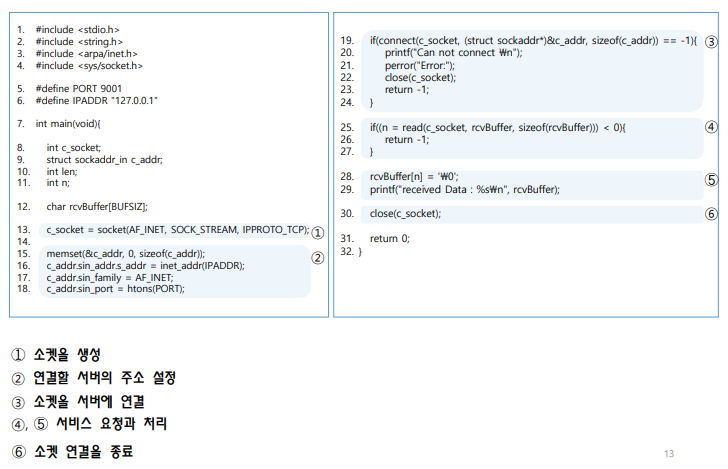
통신의 구성요소

소켓의 개념

TCP 소켓 프로그래밍

**2주차**

**Hello, world 출력 소켓 프로그램 / 클라이언트**



**socket()을 이용한 소켓 생성**

int socket(int domain, int type, int protocol);

**인터넷과 연결하기 위한 접점인 소켓(endpoint socket) 생성**

- domain: 소켓의 사용 영역을 정의

- type: 소켓 유형을 정의

- protocol: 소켓이 사용할 프로토콜을 정의

**socket 함수를 이용한 소켓 생성의 예**

1. TCP 소켓:  
   socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP)
2. UDP 소켓:  
   socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP)

**socket 함수 반환 값**

- 성공적으로 소켓을 만들면 0보다 큰 int 값을 반환

- 소켓지정번호, socket descriptor라고 부름

- 소켓을 지시하며, 이를 이용해서 소켓을 제어

s\_socket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

c\_socket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

**소켓지정번호(socket descriptor)**

descriptor

- 유닉스에서 파일을 새로 열면 int형 타입의 파일 디스크립터를 리턴

- 프로그램에서 이 파일을 액세스할 때 해당 파일 디스크럽터를 사용

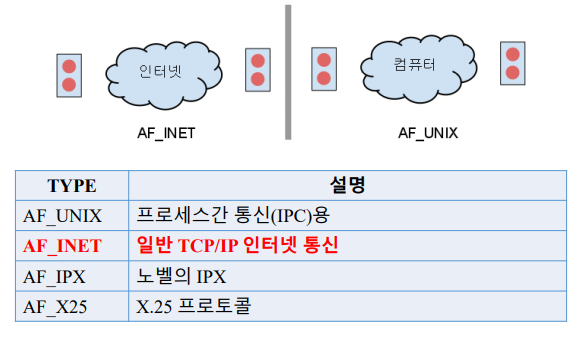
- 유닉스에서는 각종 하드웨어 장치 파이프 소켓 등을 파일로 취급

소켓지정번호는 응용 프로그램 내에서 순서대로 배정됨

- 프로그램 내에서면 유일하게 구분

**int socket(int domain, int type, int protocol);**

**- domain: 소켓이 사용되는 네트워크 영역을 정의**



**- type: 통신에 사용할 패킷의 타입을 지정**

**- protocol: 통신에 사용할 프로토콜 지정**



**connect()을 이용하여 연결하기**

int connect(int sockfd, struct sockaddr \*serv\_addr, socklen\_t addrlen);

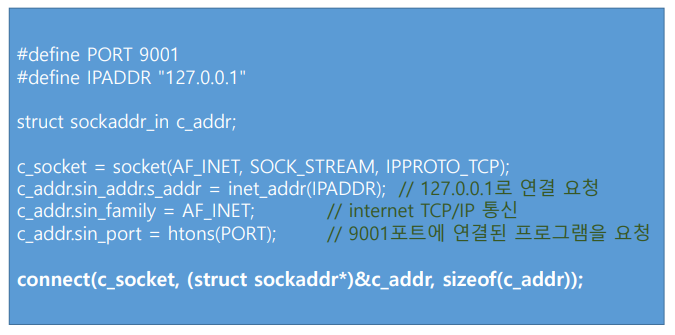
연결하고자 하는 상대 Node의 IP 주소와 Port 번호를 이용하여 소켓 연결

- sockfd: 소켓파일 지정 번호 -> 클라이언트

- serv\_addr: 연결할 서버의 IP 주소와 Port 번호를 포함한 구조체

- addrlen: 구조체 \*serv\_addr의 크기

**connect 함수를 이용한 연결 요청의 예**



**데이터 통신하기**

**데이터 쓰기**

int write(int sockfd, const void \*buf, size\_t count);

**데이터 읽기**

int read(int sockfd, void \*buf, size\_t count);

- sockfd: 소켓 지정 번호

- buf: 통신에 사용할 데이터를 가리키는 포인터

- count: 통신에 사용할 데이터의 크기

**연결 종료**

int close(int sockfd);

**소켓을 닫고 연결 종료**

- 데이터 통신이 끝났다면, close 함수를 이용해서 소켓을 닫음

- 소켓을 닫지 않을 경우 자원 누수 발생

**클라이언트 네트워크 프로그램의 흐름**

**클라이언트 프로그램의 흐름**

**- socket(): 소켓 생성**

// int socket(int domain, int type, int protocol

// c\_socket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

**- connect(): 연결 요청**

// int connect(int sockfd, struct sockaddr \*serv\_addr, socklen\_t addrlen);

// connect(c\_socket, (struct sockaddr\*)&c\_addr, sizeof(c\_addr)

**- read()/write(): 데이터 통신**

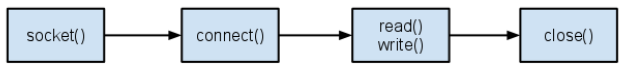
// int write(int sockfd, const void \*buf, size\_t count); // int read(int sockfd, void \*buf, size\_t count);

// read(c\_socket, rcvBuffer, sizeof(rcvBuffer))

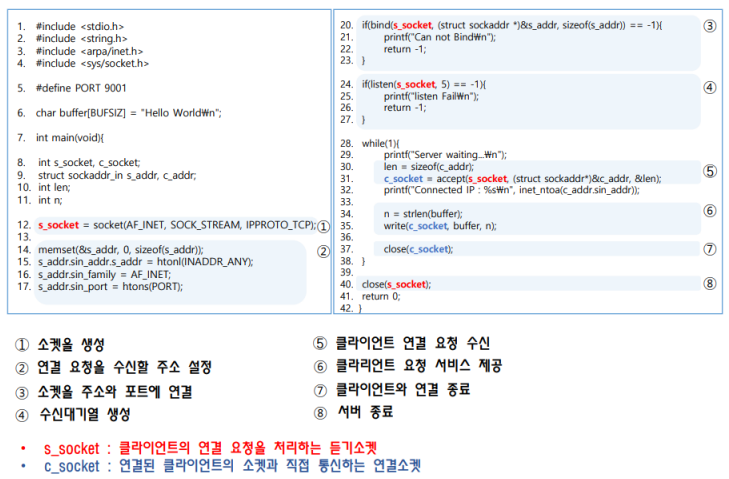
**- close(): 소켓 닫기**

// int close(int sockfd)

// close(c\_socket);



**Hello, world 출력 소켓 프로그램 / 서버**

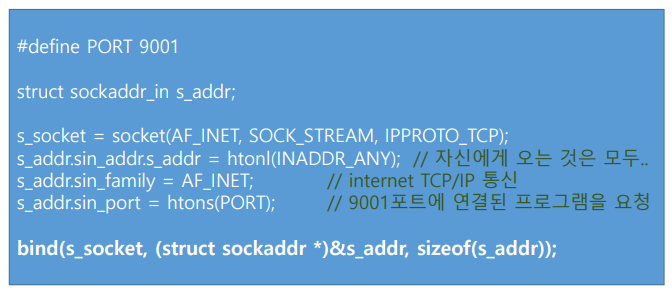


**서버 프로그램 만들기 – bind()**

**bind 함수**

- 소켓을 인터넷 주소(IP 주소 & Port 번호)에 묶어줌

int bind(int sockfd, struct sockaddr \*my\_addr, socklen\_t addrlen);



**서버 프로그램 만들기 – listen()**

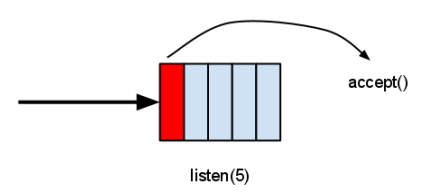
**listen 함수: 수신 대기열 생성**

- 클라이언트의 요청은 먼저 수신 대기열에 들어감

int listen(int sockfd, int backlog);

- sockfd: 소켓 지정 번호

- backlog: 연결 대기열의 크기



**서버 프로그램 만들기 – accept()**

**accept 함수**

- 수신 대기열의 맨 앞에 있는 클라이언트 요청을 읽음

- 클라이언트와의 통신을 담당할 소켓 지정 번호를 반환

int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen)

- sockfd: 소켓 지정 번호

- \*addr: 연결 요청을 한 클라이언트의 소켓 주소 구조체

- \*addrlen: \*addr 구조체 크기의 포인터

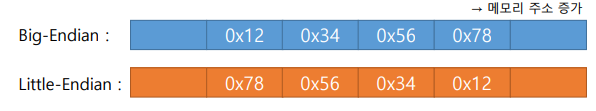
**기타 함수**



**컴퓨터마다 메모리를 데이터에 저장하는 방식이 다름**

- 빅엔디안(Big-Endian), 리틀엔디안(Little-Endian)

- 예: 0x12345678 저장 시



**호스트 바이트 순서**

- 컴퓨터가 내부 메모리에 숫자를 저장하는 순서

- CPU 종류에 따라 빅엔디안 또는 리틀엔디안 사용

**네트워크 바이트 순서**

- 포트번호나 주소와 같은 정보를 바이트 단위로 네트워크로 전송하는 순서

- 빅엔디안 사용

**바이트 순서가 바뀌는 문제의 해결방법**

- 네트워크로 전송하기 전에 htons()함수를 사용하여 네트워크 바이트 순서로 바꿈

- 네트워크로부터 수신한 숫자는 ntohs()함수를 사용하여 자신의 호스트 바이트 순서로 바꿈

cf) htonl() / ntohl()

기본적으로 PC는 리틀엔디안으로 되어 있기 때문에 이를 빅엔디안으로 변경하는 방법은 다음과 같다

**IP의 경우: htonl(ip);**

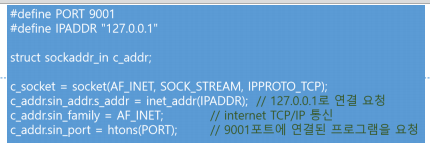
**포트의 경우: htons(port);**

반대의 경우는

**IP의 경우: ntohl(ip);**

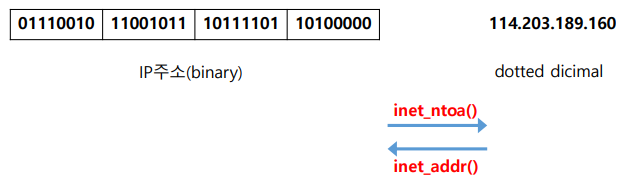
**포트의 경우: ntohs(port);**

**기타 함수**



**IP 주소변환**

- 4바이트 IP주소를 dotted decimal 방식으로 상호 변환 가능



**- inet\_ntoa(): 바이너리 IP주소를 dotted decimal 형태의 IP주소로**

**- inet\_addr(): dotted decimal 형태의 IP주소를 바이너리 IP주소로**

**서버 네트워크 프로그램의 흐름**

**서버 프로그램의 흐름**

- socket(): 소켓 생성

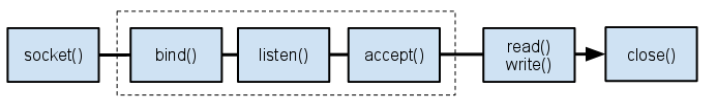
- bind(): 소켓을 인터넷 주소와 포트 번호에 묶음

- listen(): 수신 대기열 생성 (listen queue)

- accept(): 연결 대기

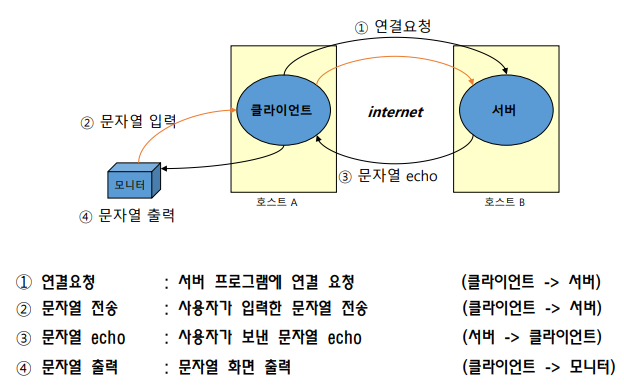
- read()/write(): 데이터 통신

- close(): 소켓 닫기

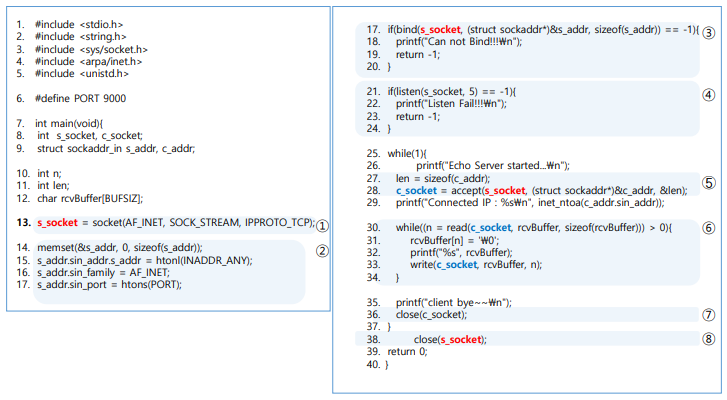


**3주차**

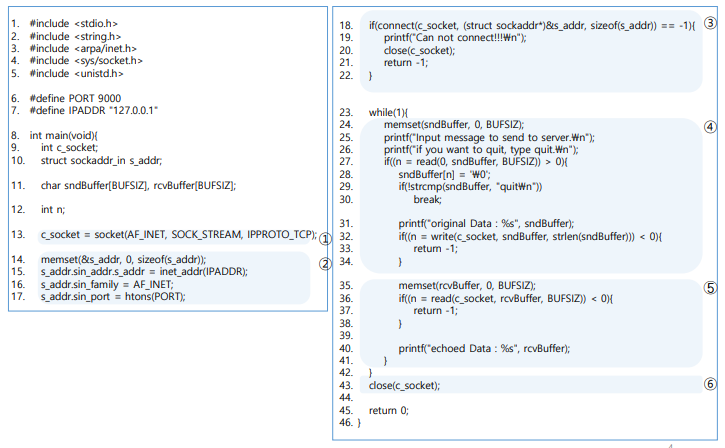
**echo 프로그램**



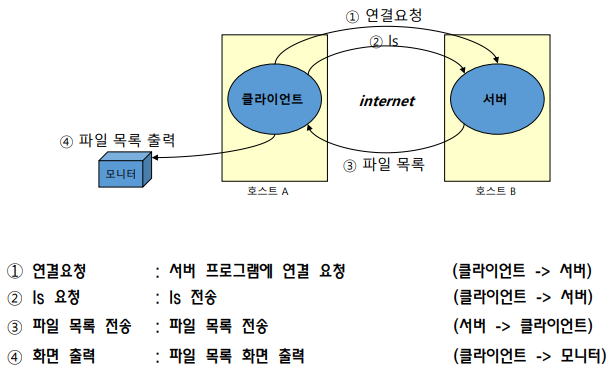
**echo 프로그램 / 서버**



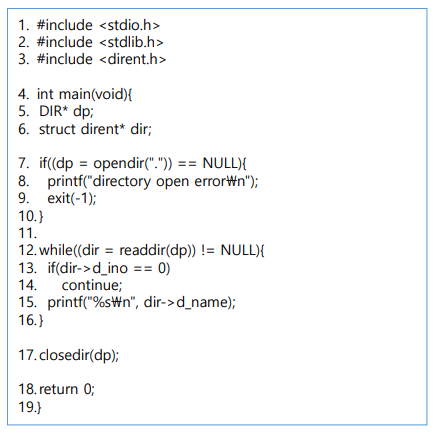
**echo 프로그램 / 클라이언트**



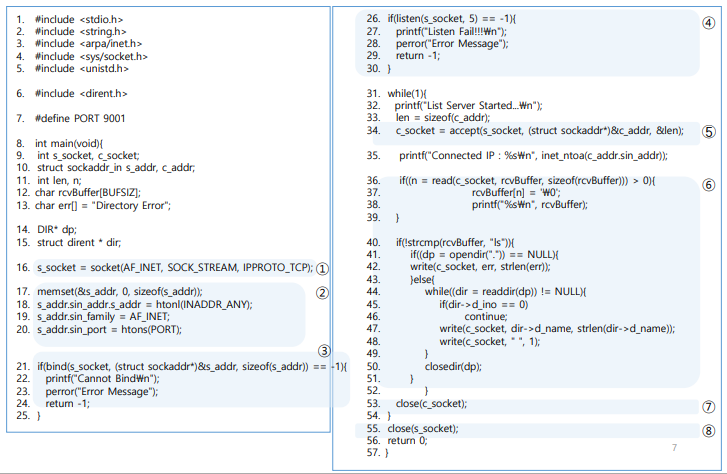
**서버 디렉토리의 파일 목록 출력**



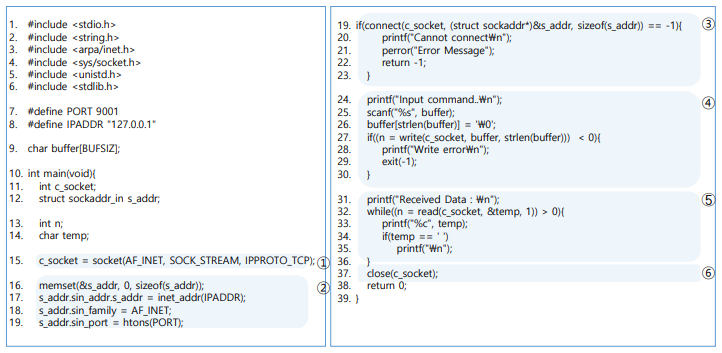
**디렉토리의 파일 목록 출력**



**서버 디렉토리의 파일 목록 출력 / 서버**



**서버 디렉토리의 파일 목록 출력 / 클라이언트**



**정리**

echo 프로그램

서버 디렉토리 파일 목록 출력 프로그램

\* 구조는 변하지 않고 내용(처리) 부분만 변화됨

- 서버: socket(), bind(), listen(), accept(), read()/write(), close()

- 클라이언트: socket(), connect(), read()/write(), close()

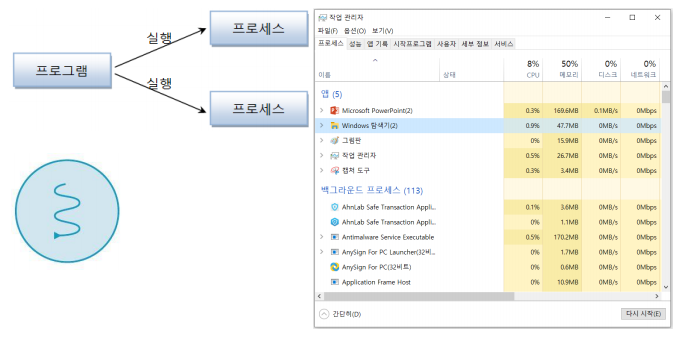
**4주차**

**프로세스와 스레드**

프로세스(process)

- 실행 중인 하나의 프로그램

- 하나의 프로그램이 다중 프로세스를 만들기도 함



**스레드(thread)**

- 프로세스 내부에 있는 제어의 단을 순차 흐름

- 하나의 스레드는 프로세스와 같이 시작, 실행, 종료의 순서를 가짐

- 스레드 자체는 프로세스가 아니므로 홀로 실행될 수 없음

・프로세스 내에서만 실행 가능

**멀티 태스킹(multi tasking)**

- 두 가지 이상의 작업을 동시에 처리하는 것

1. 멀티 프로세스

독립적으로 프로세스들을 실행하고 여러 가지 작업 처리

2. 멀티 스레드

한 개의 프로세스를 실행하고 내부적으로 여러 작업 처리

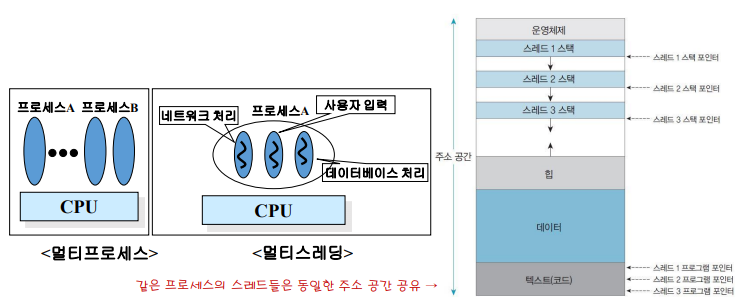
**스레드의 장점**

- 완전한 프로세스의 상태를 저장하는 값비싼 부담을 덜 수 있음

・문맥 전환이 동일 주소 공간 내에서 행해지기 때문

・스레드 간의 문맥 전환에서는 단지 소수의 레지스터, 스택 포인터, 프로그램 카운터 등에 대해서만 상태 저장과 복구가 수행

- 프로그램의 실행 시간을 단축시킴



**메인(main) 스레드**

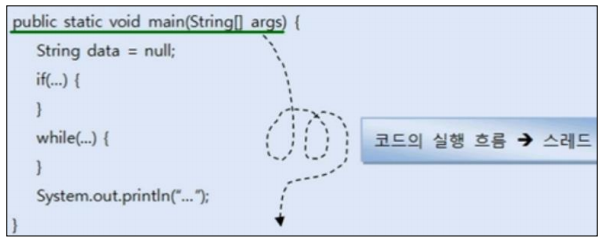
- 모든 자바 프로그램은 메인 스레드가 main() 메소드 실행하며 시작

- main() 메소드의 첫 코드부터 아래로 순자적으로 실행

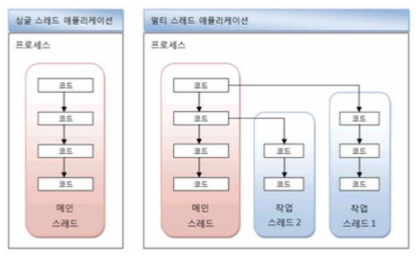
- 실행 종료 조건

・마지막 코드 실행

・return문 실행



- main 스레드는 작업 스레드들을 만들어 병렬로 코드들을 실행



- 프로세스의 종료

・싱글 스레드: 메인 스레드가 종료하면 프로세스도 종료

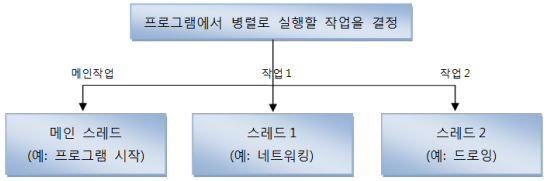
・멀티 스레드: 실행 중인 스레드가 하나라도 있다면, 프로세스 미종료

・메인 스레드가 작업 스레드보다 먼저 종료되더라도 작업 스레드가 계속 실행 중이라면 프로세스는 종료되지 않음

**작업 스레드 생성과 실행**

**멀티 스레드로 실행하는 어플리케이션 개발**

- 몇 개의 작업을 병렬로 실행할지 결정하는 것이 중요



**작업 스레드 생성 방법**

1. Thread 클래스로부터 직접 생성  
   Runnable을 매개값으로 갖는 생성자 호출
2. Thread 하위 클래스로부터 생성  
   Thread 클래스 상속 후 run 메소드 재정의 해 스레드가 실행할 코드 작성

**프로그램 이해를 위해 필요한 개념들..**

**클래스**

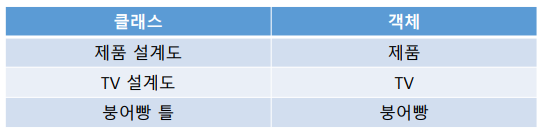
- 정의: 객체를 정의해 놓은 것

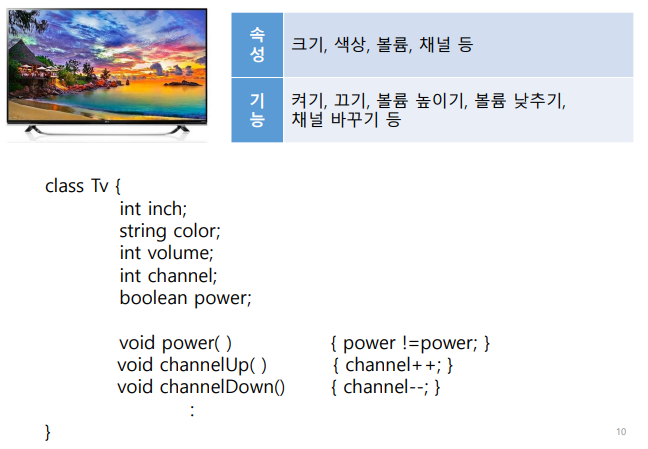
- 용도: 객체를 생성하는 데 사용

**객체**

- 정의: 실제로 존재하는 것. 사물 또는 개념

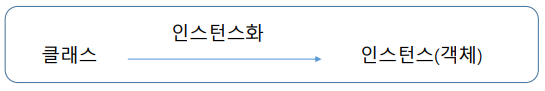
- 용도: 객체가 가지고 있는 기능과 속성에 따라 다름



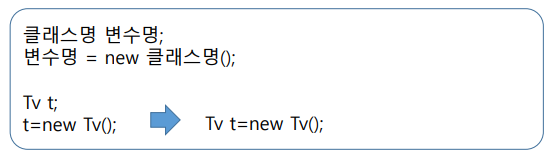


**인스턴스화**

- 클래스로부터 객체를 만드는 과정



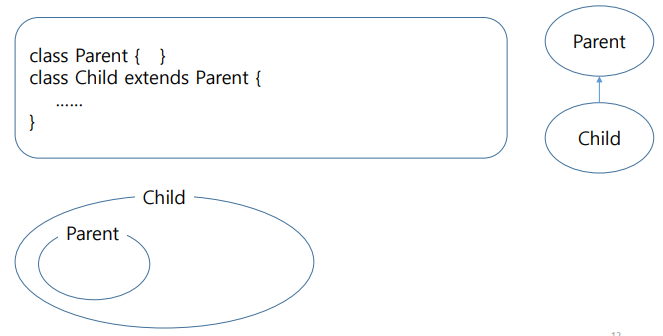
**객체의 생성과 사용**



**상속**

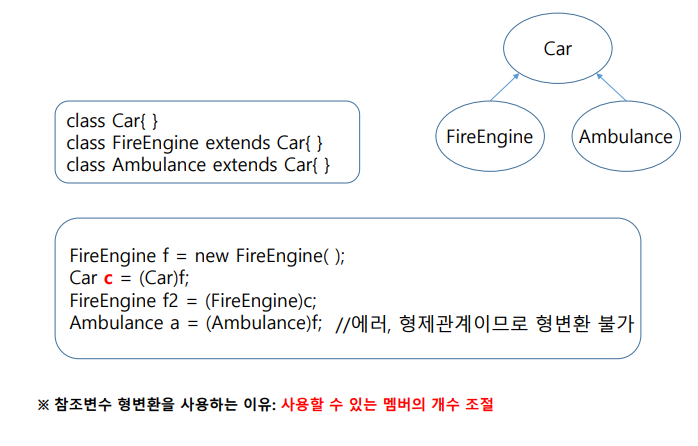
- 기존 클래스를 재사용하여 새로운 클래스를 작성하는 것

- 재사용성이 높고 코드의 중복제거에 용이



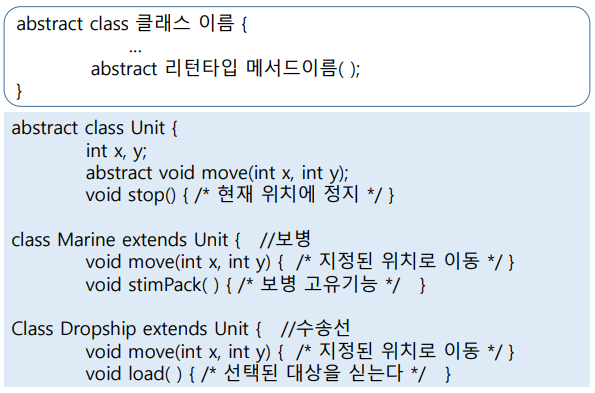
**창조변수의 형변환**

- 상속관계에 있는 클래스 사이에서만 가능



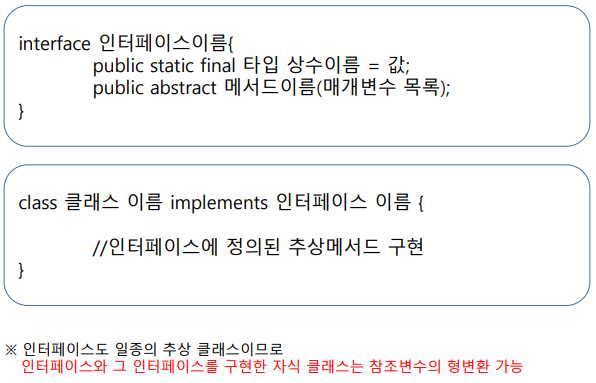
**추상 클래스**

- 추상 메서드를 포함하고 있는 클래스

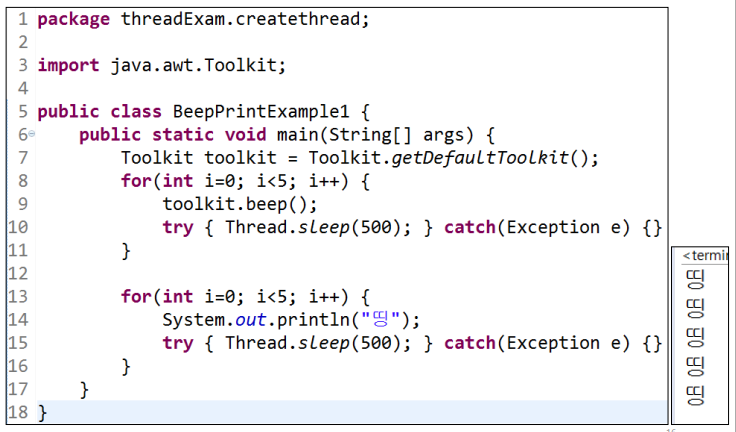


**인터페이스**

- 일종의 추상클래스

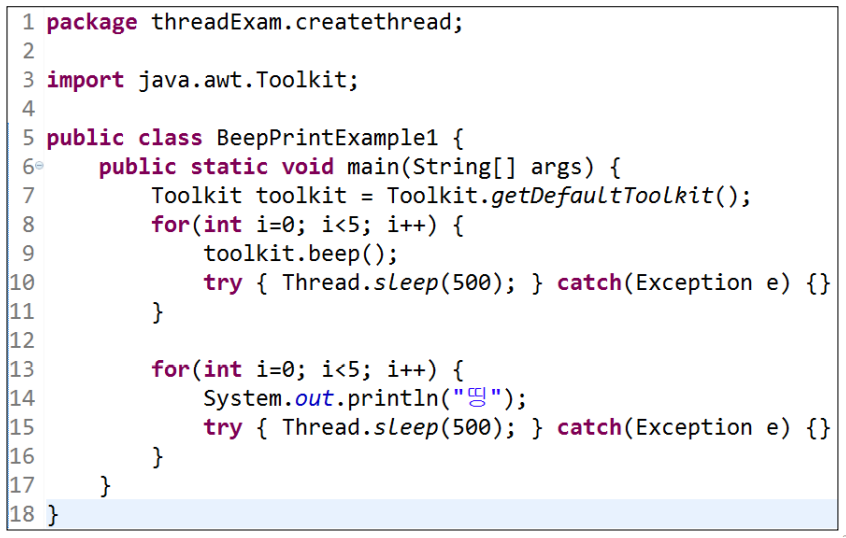


**두 가지 작업을 처리하는 예제**

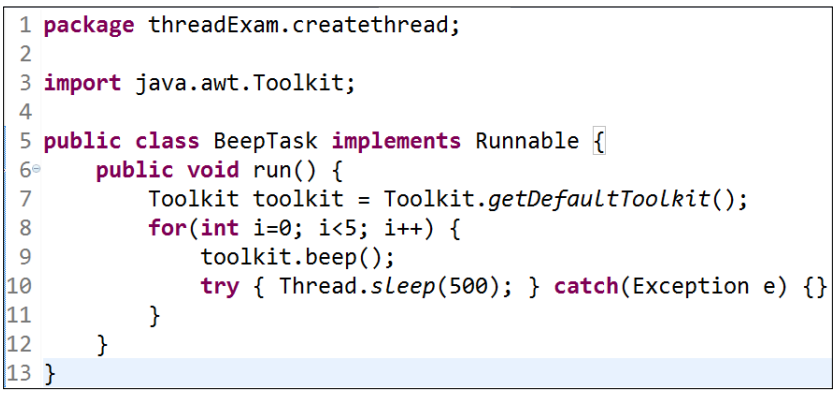


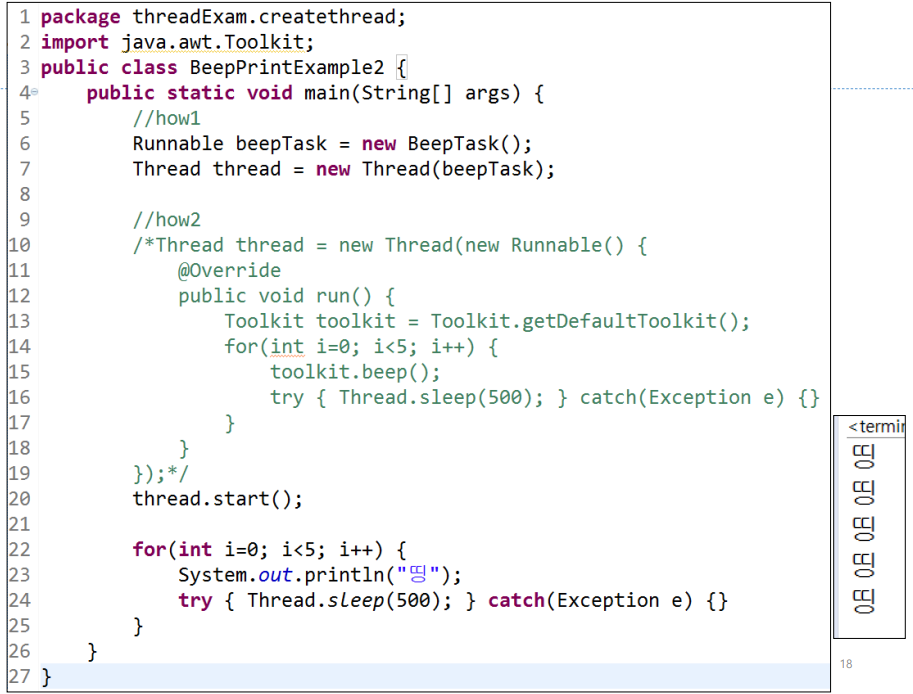
**5주차**

**작업 스레드 생성과 실행**

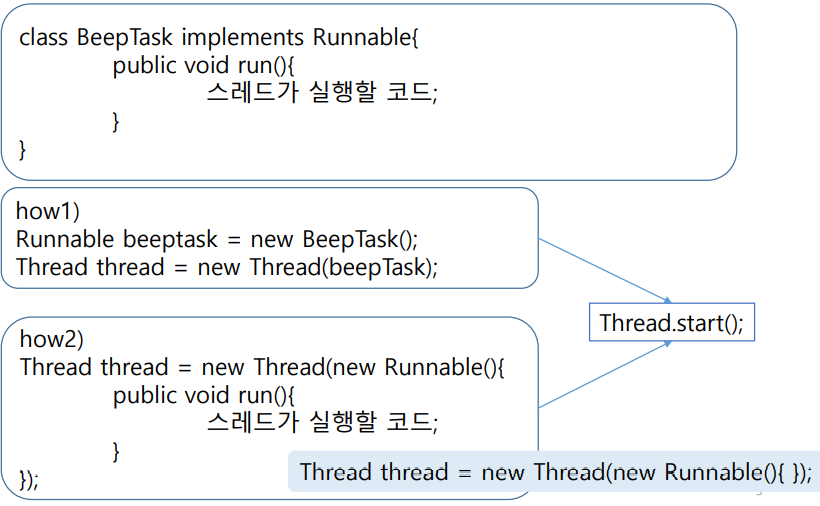


**1. Thread 클래스로부터 직접 생성**

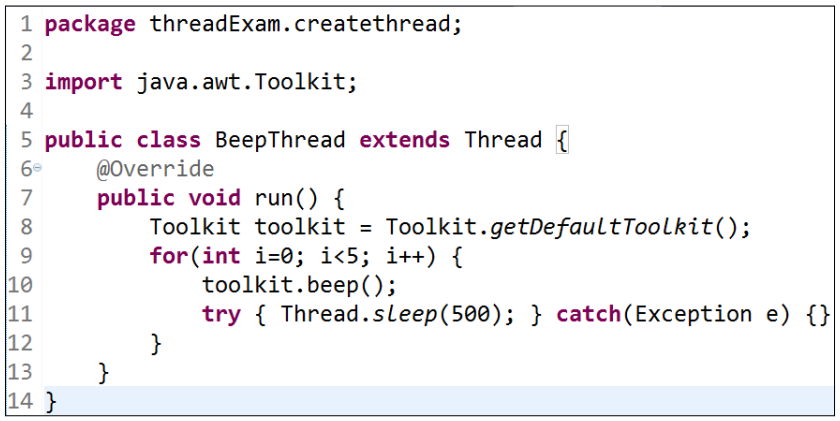


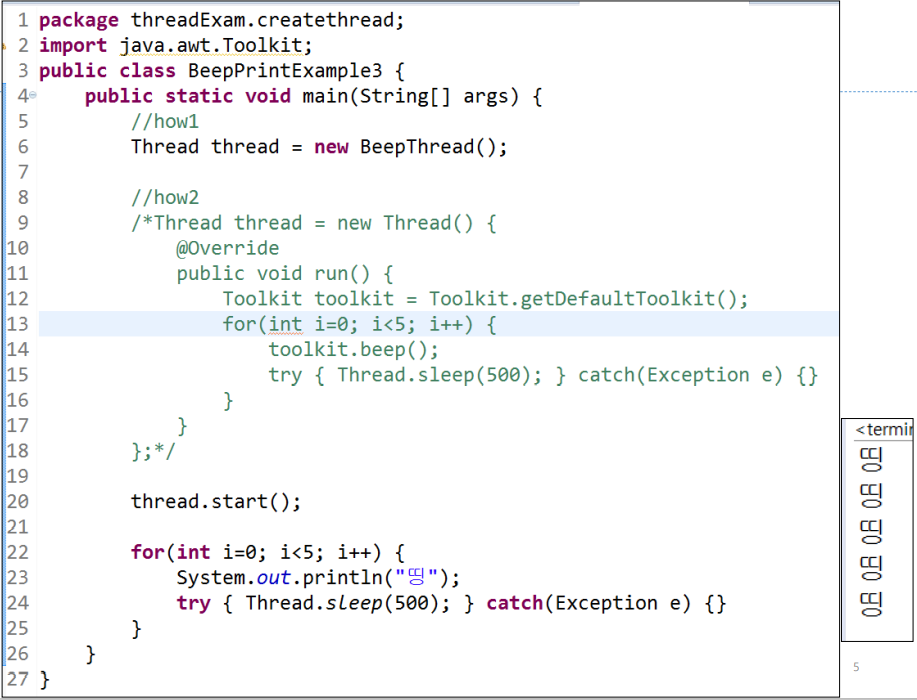






**2. Thread 하위 클래스로부터 생성**

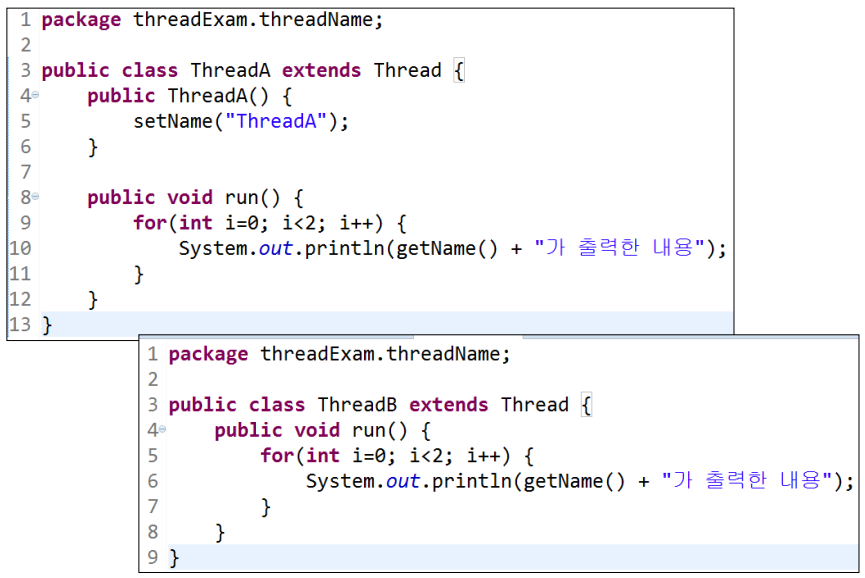


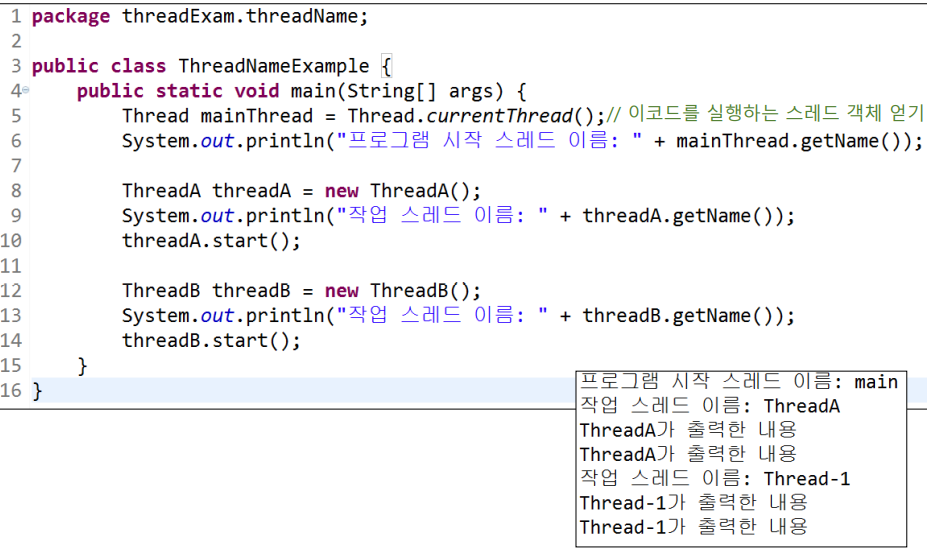


**스레드의 이름**

* 메인 스레드 이름: main
* 작업 스레드 이름 자동 설정): Thread-n  
  
* 작업 스레드 이름 변경  
  
* 코드 실행하는 현재 스레드 객체의 참조 얻기  
  

**스레드 이름 예제**

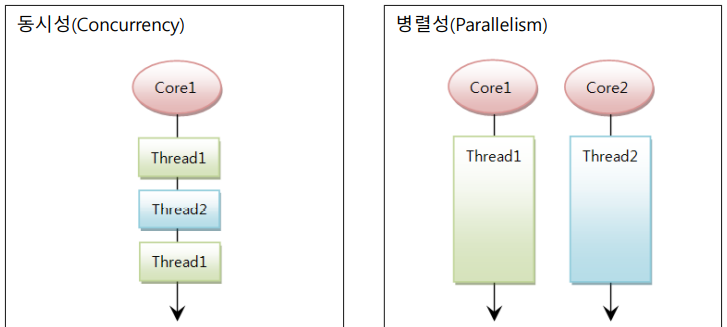




**스레드 우선 순위**

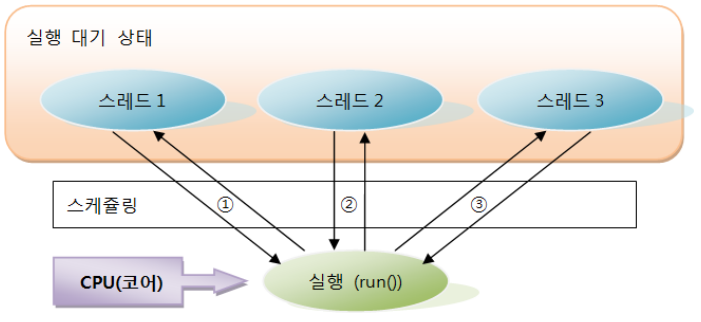
**동시성과 병렬성**

* 동시성(Concurrency)
  + 멀티 작업을 위해 하나의 코어에서 멀티 스레드가 번갈아 가며 실행하는 성질
* 병렬성(Parallelism)
  + 멀티 작업을 위해 멀티 코어에서 개별 스레드를 동시에 실행하는 성질



**스레드 스케줄링**

* 스레드의 개수가 코어의 수보다 많을 경우
  + 스레드를 어떤 순서로 동시성으로 실행할 것인가 결정 -> 스레드 스케줄링
  + 스케줄링 의해 스레드들은 번갈아 가며 run() 메소드를 조금씩 진행



**자바의 스레드 스케줄링**

* 우선 순위(Priority) 방식과 순환 할당(Round-Robin) 방식 사용

1. 우선 순위 방식 (코드로 제어 가능)

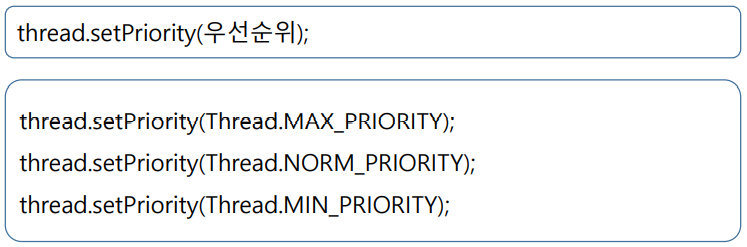
・우선 순위가 높은 스레드가 실행 상태를 더 많이 가지도록 스케줄링

・1~10까지 값을 가질 수 있으며 기본은 5

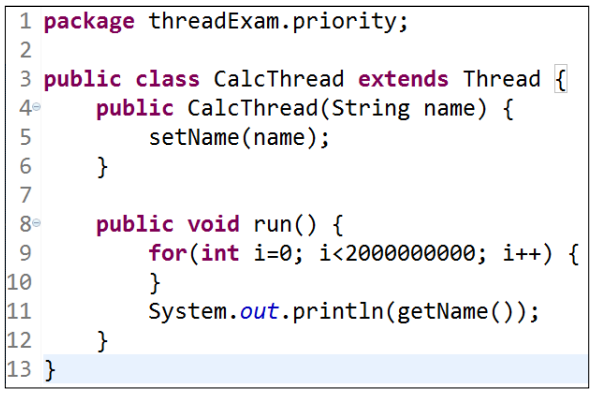
2. 순환 할당 방식 (코드로 제어할 수 없음)

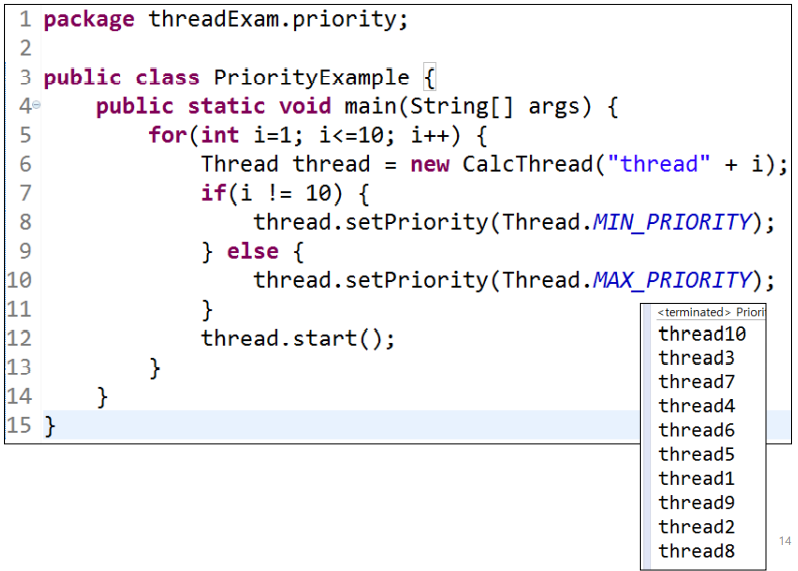
・시간 할당량(Time Slice) 정해서 하나의 스레드를 정해진 시간만큼 실행

**스레드 우선 순위**

* 스레드들이 동시성을 가질 경우 우선적으로 실행할 수 있는 순위
* 우선순위는 1(낮음)에서부터 10(높은)까지로 부여
  + 모든 스레드들은 기본적으로 5의 우선 순위를 할당
* 우선 순위 변경 방법  
  

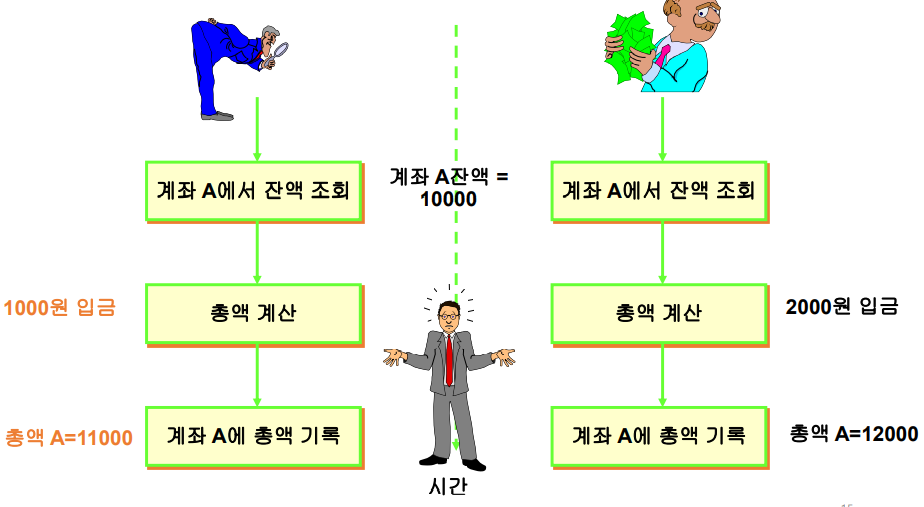
**스레드 우선 순위 예제**





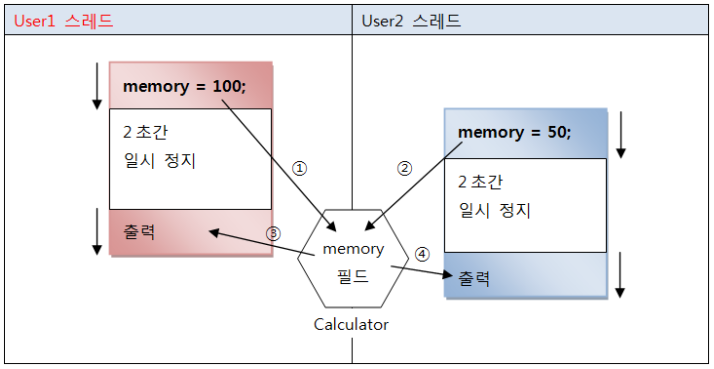
**동기화 메소드와 동기화 블록**

**동기화란?**

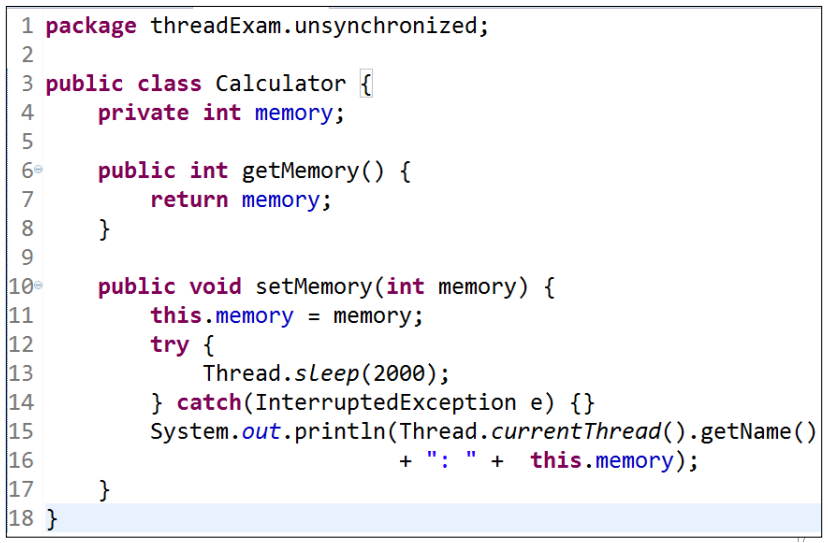


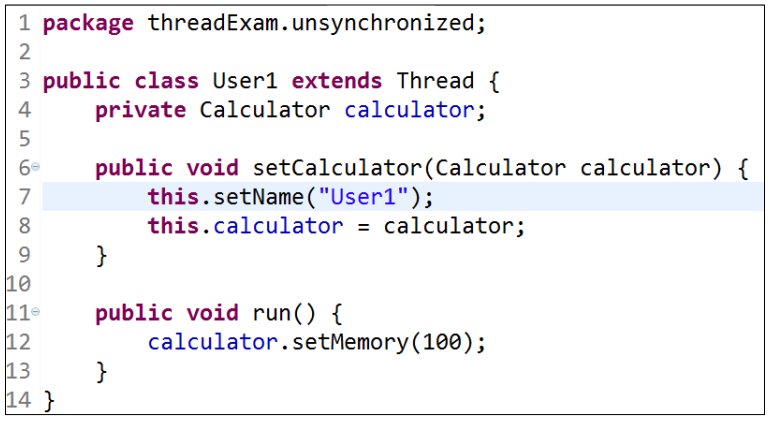
**공유 객체를 사용할 때의 주의할 점**

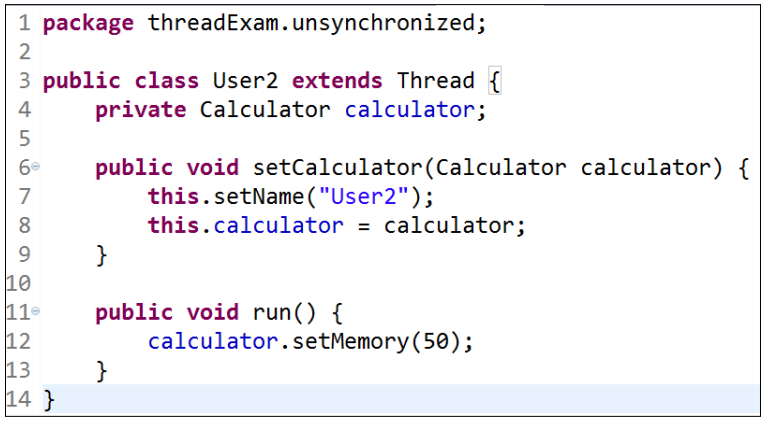
* 멀티 스레드가 하나의 객체를 공유해서 생기는 오류

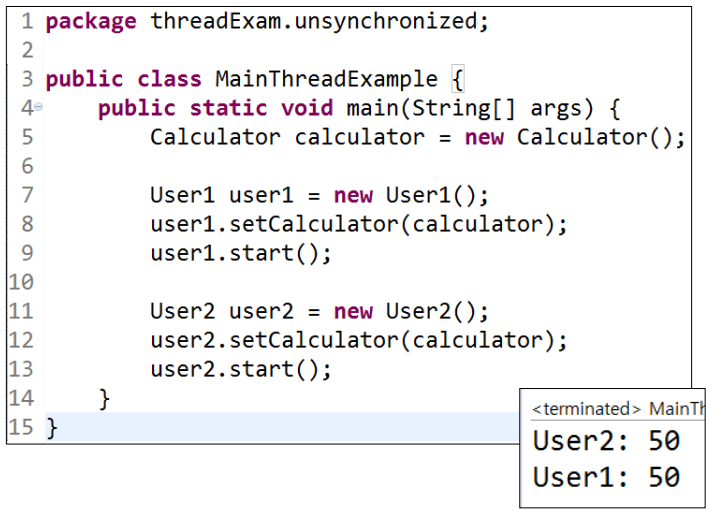


**공유객체 사용시 문제점 예제**

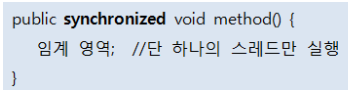
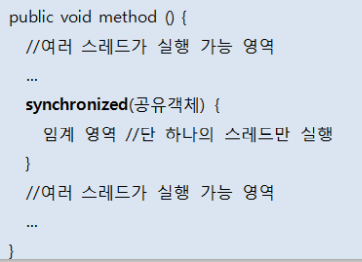








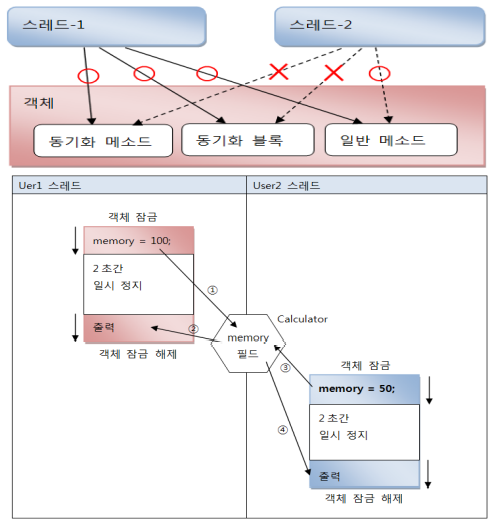
**동기화 메소드 및 동기화 블록 – synchronized**

* 단 하나의 스레드만 실행할 수 있는 메소드 또는 블록
* 다른 스레드는 메소드나 블록이 실행이 끝날 때까지 대기해야 함
* 동기화 메소드  
  
* 동기화 블록  
  

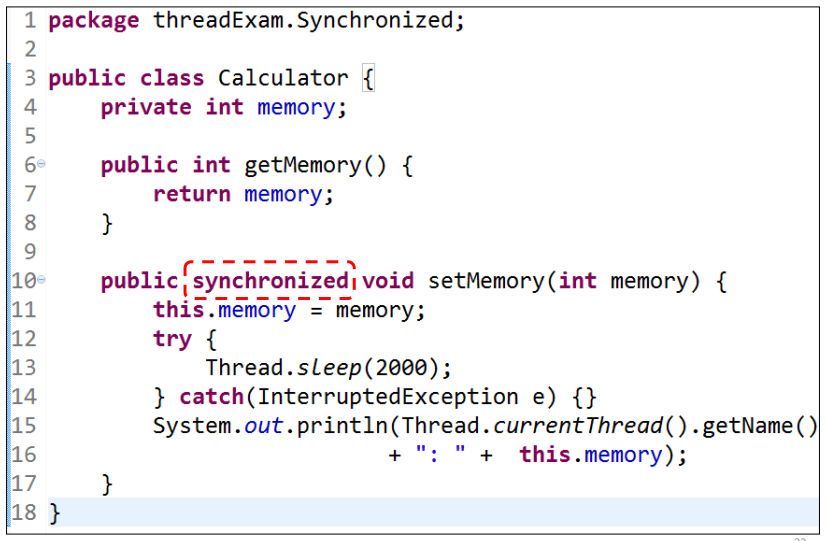
**동기화 메소드**

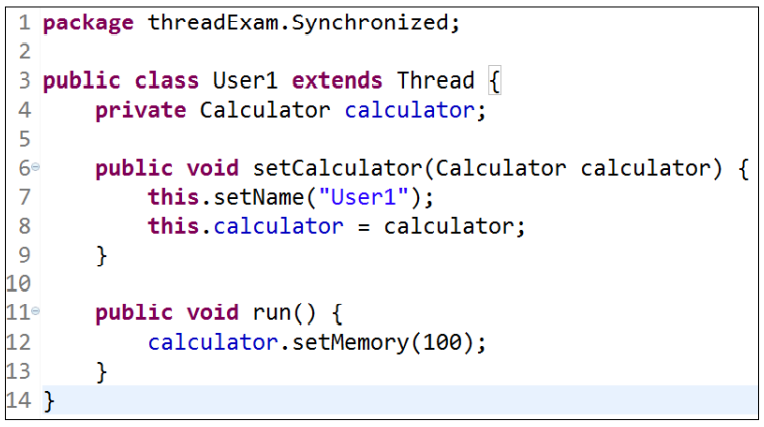
* 동일한 객체에 대하여 여러 스레드의 중첩 실행을 방지할 때 사용하는 메소드
  + 동일한 객체에 대하여 한 스레드가 동기화 메소드를 실행하고 있으면, 다른 스레드는 그 메소드를 실행할 수 없음
* 자바에서의 동기화 메소드 사용
  + “synchronized”로 선언
  + 스레드가 어떤 객체의 동기화 메소드를 호출하면 그 객체는 lock됨
  + 또 다른 스레드가 동일 객체의 동기화 메소드를 호출하면, 그 스레드는 lock이 해제될 때까지 블록됨  
    => “상호 배제”

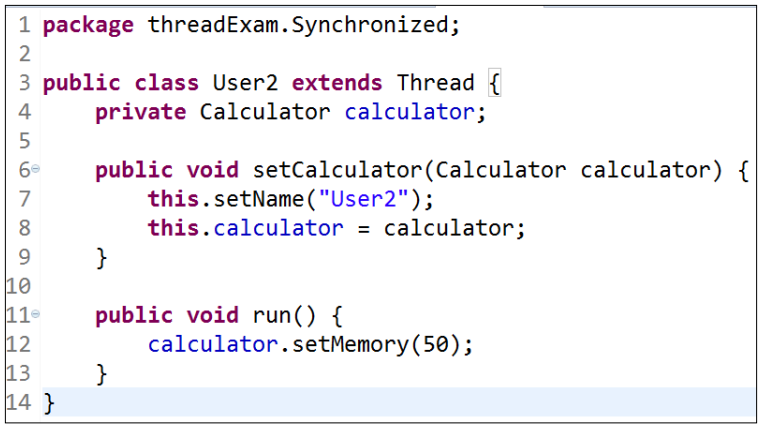
**동기화 메소드 및 동기화 블록**

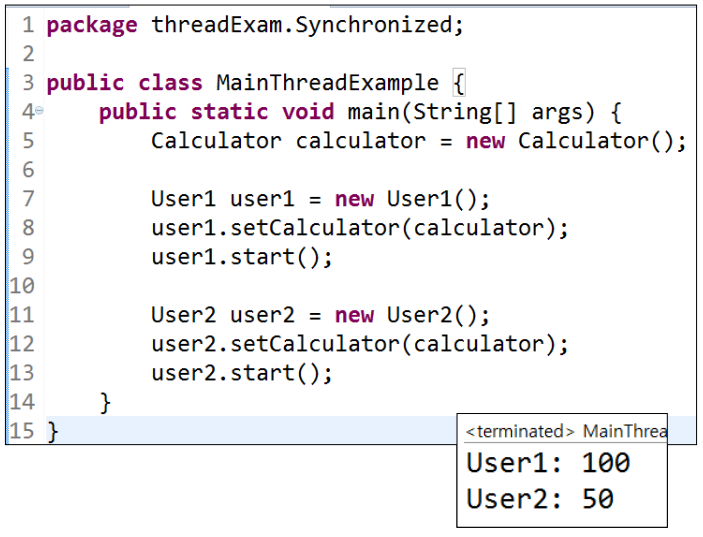


**동기화 예제 1**

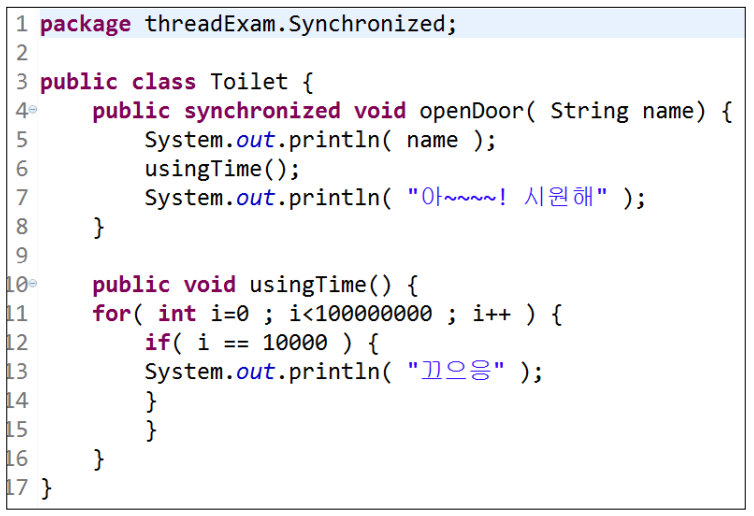


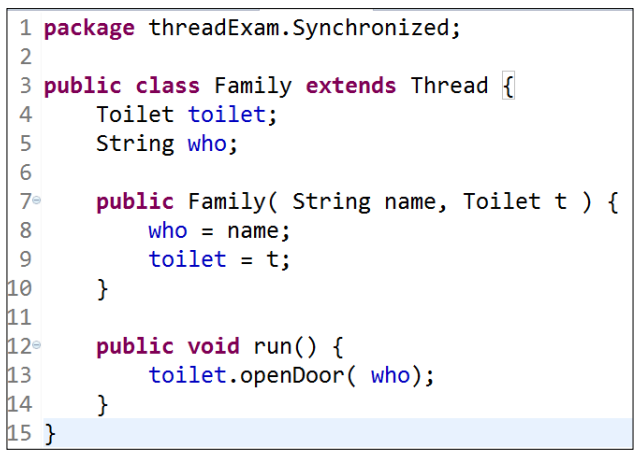


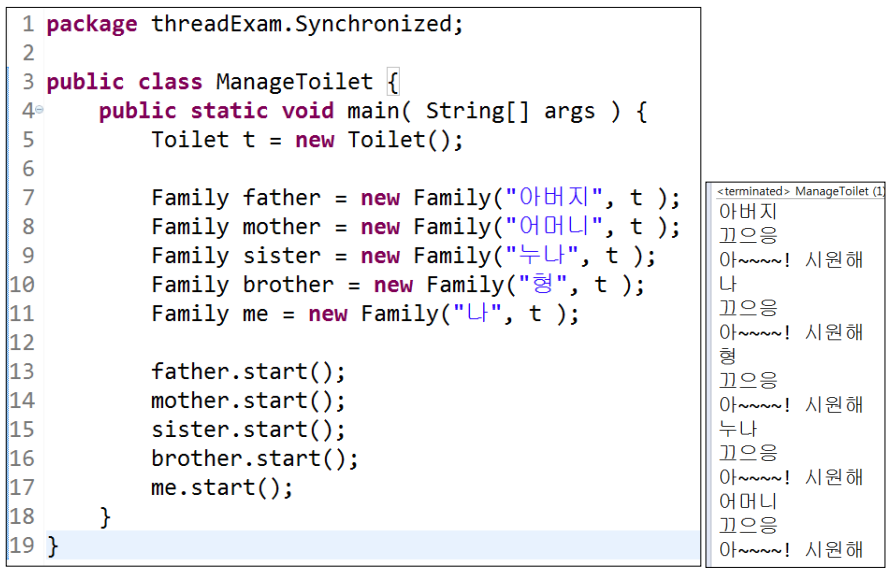




**동기화 예제 2**







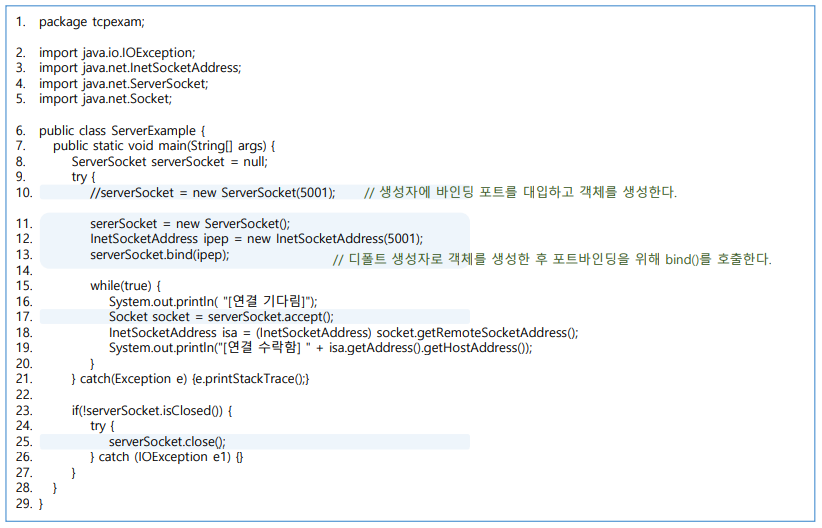
**6주차**

**자바 소켓 통신**

**ServerSocket과 Socket**

* ServerSocket: 소켓간의 연결만 처리
* Socket: 프로세스간의 실제 통신을 담당  
  InputStream과 OutStream을 가지고 있음

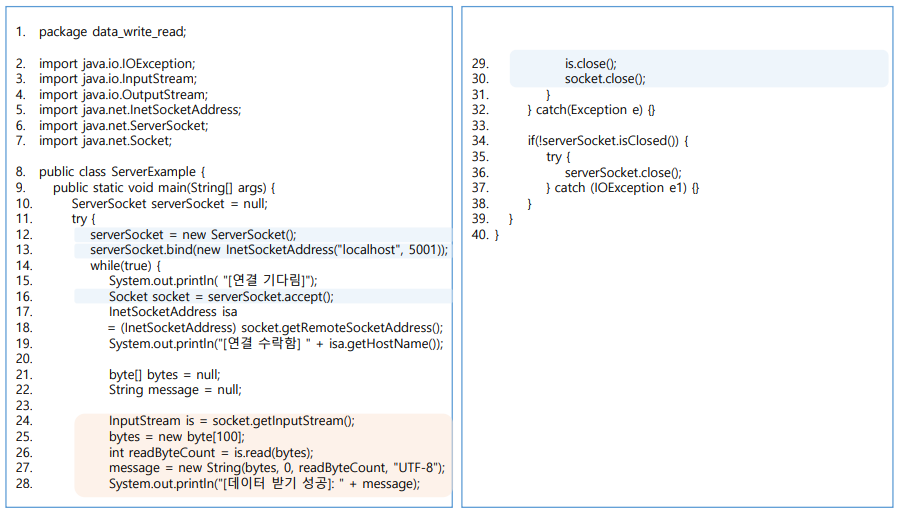
**자바 소켓 통신 예제(서버\_1)**



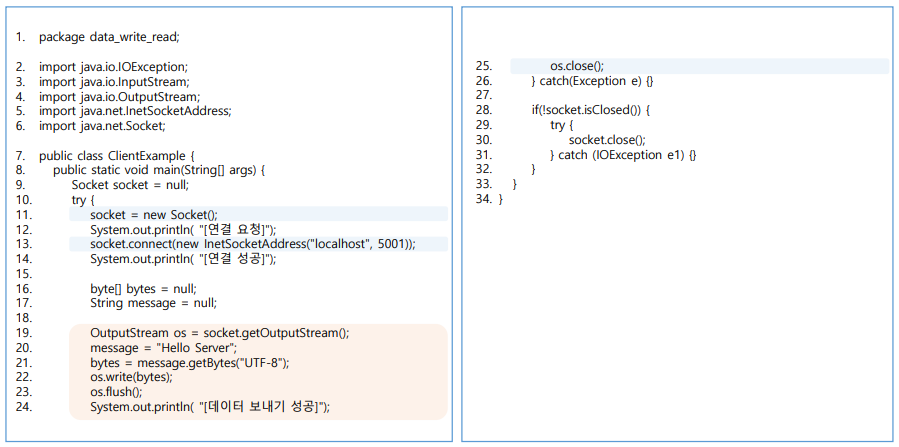
**자바 소켓 통신 예제(클라이언트\_1)**



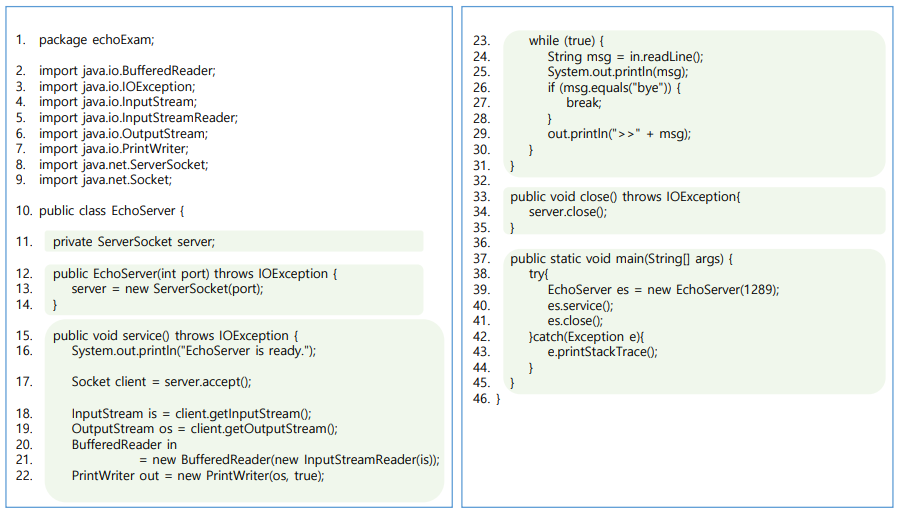
**자바 소켓 통신 예제(서버\_2)**



**자바 소켓 통신 예제(클라이언트\_2)**



**자바 소켓 통신 예제(서버\_3)**



**자바 소켓 통신 예제(클라이언트\_3)**

