



Java Network Programming & Web

강사 소개



- 문성훈
- [주] ATGLab. 개발 / 컨설팅 / 직무교육
Gamification 전문기업.
Sencha Inc. 국제 공인 Trainer.
W3C(W3DevCampus Online Trainer.)
- 고려대학교 컴퓨터 학과 Ph.D.
- 삼성전자 / LGCNS / KT 직무교육
한국IBM / SK C&C 직무교육
CJ 올리브 네트워크 직무교육
LGCNS 신입사원교육 컨설팅
고려대학교 / 한국기술교육대학교 강의
- Email : moon9342@atglab.co.kr

- Java Thread의 이해
- Java IO의 이해
- TCP/UDP Programming & Web Programming



- 이론 / 실습 비율

- 40% / 60%

- 강의 시간

- 45분 강의 / 15분 휴식 (09:00 ~ 18:00)

- 점심 시간

- 12시 30분 ~ 13시 30분



- 개발 Tool download

- JDK  <http://java.oracle.com>
- Eclipse  <http://www.eclipse.org/>



Thread



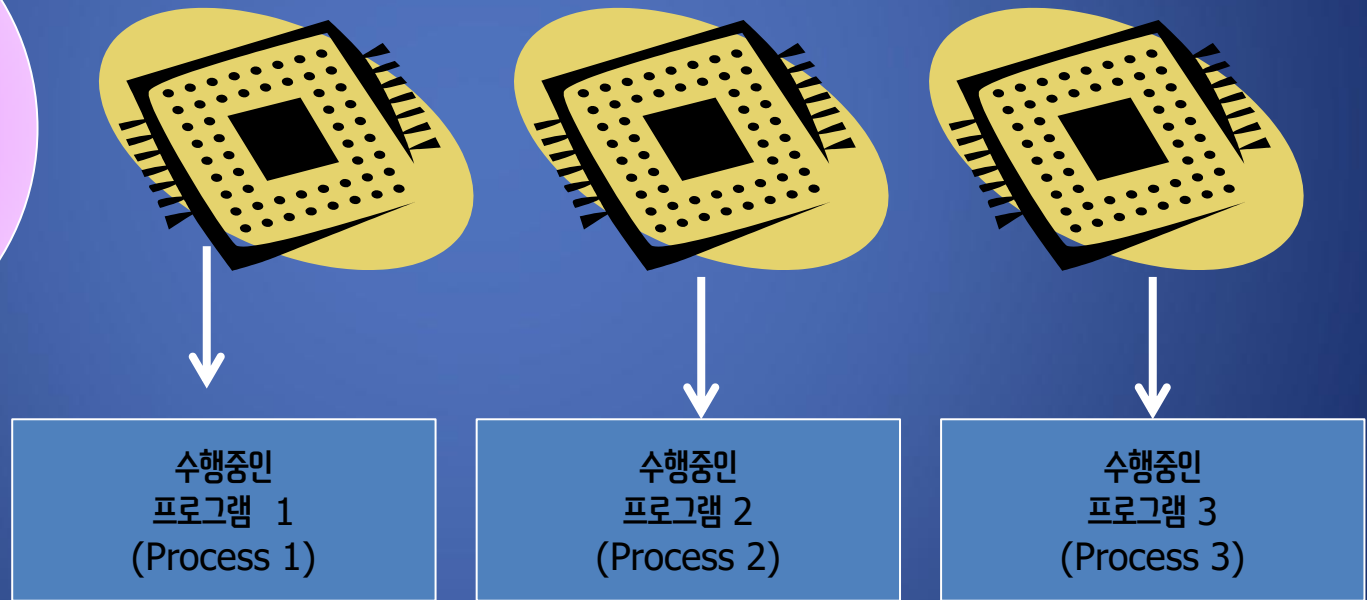
Multi
Processing

Multi
Tasking

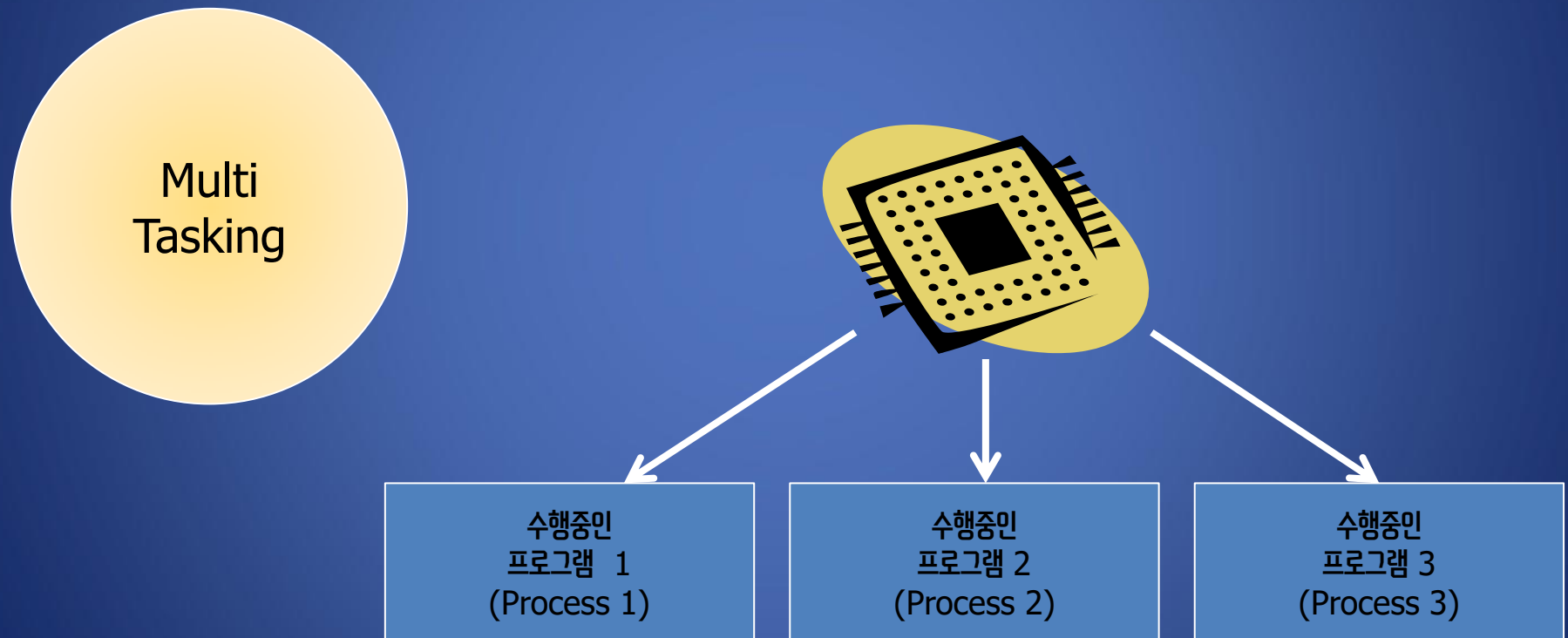
Multi
Threading

Thread

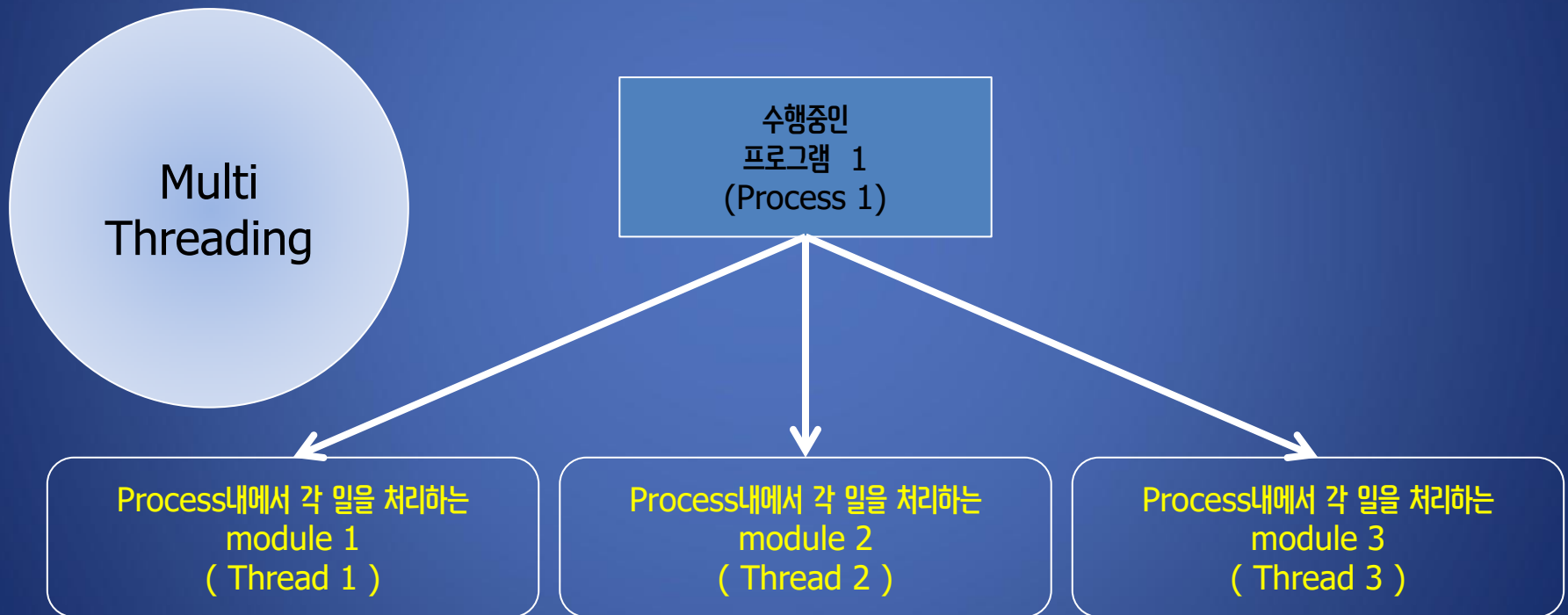
Multi
Processing



Thread



Thread



Thread

- 운영체제에서 실행되는 하나의 프로그램을 프로세스라고 볼 때 Thread는 그 프로세스 안에서 실행되는 하나의 실행흐름.
- Java는 언어적 차원에서 Thread를 지원하는 가장 일반적인 언어이며 keyword를 이용한 동기화를 지원.
- 따라서 Multi Thread 프로그램을 비교적 쉽고 명료하게 작성 가능.
- Thread의 이점과 한계.

- 우리가 사용하는 Eclipse 프로그램 내에서도 Thread가 동작하고 있다.
 - 오류 표시 기능
- Thread는 동시에 실행될 수 있는 또 다른 실행흐름을 갖게 함으로써 **효율적인 작업**을 하게 해 준다.
- 일반적으로 서버 프로그램의 경우 성능과 효율을 향상 시킬 수 있다.

Java에서 Thread는
어떻게 표현되는가?

Thread
instance

Thread class를 만들려면
어떻게 해야 하는가?

Thread class
상속

Runnable
interface 구현

Thread

- Thread 구현의 첫 번째 방법은 Thread class를 상속
- Business logic은 run() method override.
- 예제 코드 작성.

Thread

- Thread 구현의 두 번째 방법은 interface를 이용한 객체 주입이다.
- Runnable interface를 구현한 후 run() method override.
- 예제 코드 작성.

- 상속 vs. 객체 주입

- 일반적으로 class를 상속하는 것보다는 객체 주입(객체 합성)방식이 더 나은 방법
- 상속을 통한 재사용방식 : white-box reuse
- Super class가 sub class에 불필요하게 너무 많은 부분이 노출
결국, encapsulation 개념에 위배
- Sub class가 super class에 종속
- Super class의 변경이 생길 경우 sub class도 변경될 수 있다는 문제점.

- 상속 vs. 객체 주입

- 객체 주입 방식은 객체가 다른 객체의 참조자를 얻는 방식으로 runtime시에 동적으로 이루어짐.
- 보통 객체의 기능을 이용하기 위해서 interface type을 이용하므로 class의 캡슐화를 잘 유지할 수 있다.
- 참조 객체의 내부 구조를 볼 수 없기 때문에 black-box reuse라고 함.
- 객체 주입 방식은 객체 간의 관계가 수직관계가 아닌 수평 관계.
- 따라서 큰 시스템의 많은 부분에서 주입이 사용될 경우 코드의 readability가 떨어지고 이해하기 어려워지는 단점.

- Thread의 종료

- 만약 Thread를 시작 시킨 이후에 Thread를 중간에 중지시키고 싶다면?
- `interrupt()` method 이용
- 만약, `interrupt()` 호출하는 시점에 `sleep()`이 호출된 경우에는 `InterruptedException`이 발생
- 예제 코드 작성.

- Daemon Thread

- Java에서는 프로그램 내부의 모든 Thread가 종료되지 않으면 JVM이 종료되지 않는다.
- 하지만, 상황에 따라서 분리된 thread로 백그라운드 작업을 해야 하는 경우가 있다.
- JVM안에서 동작하는 Garbage Collector가 대표적인 예.
- 만약 우리가 실행하는 Java 프로그램 내에서 이런 백그라운드 작업이 일반 Thread로 설정되어 있다면 강제 종료하지 않는 한 프로그램은 종료되지 않는다.

- Daemon Thread

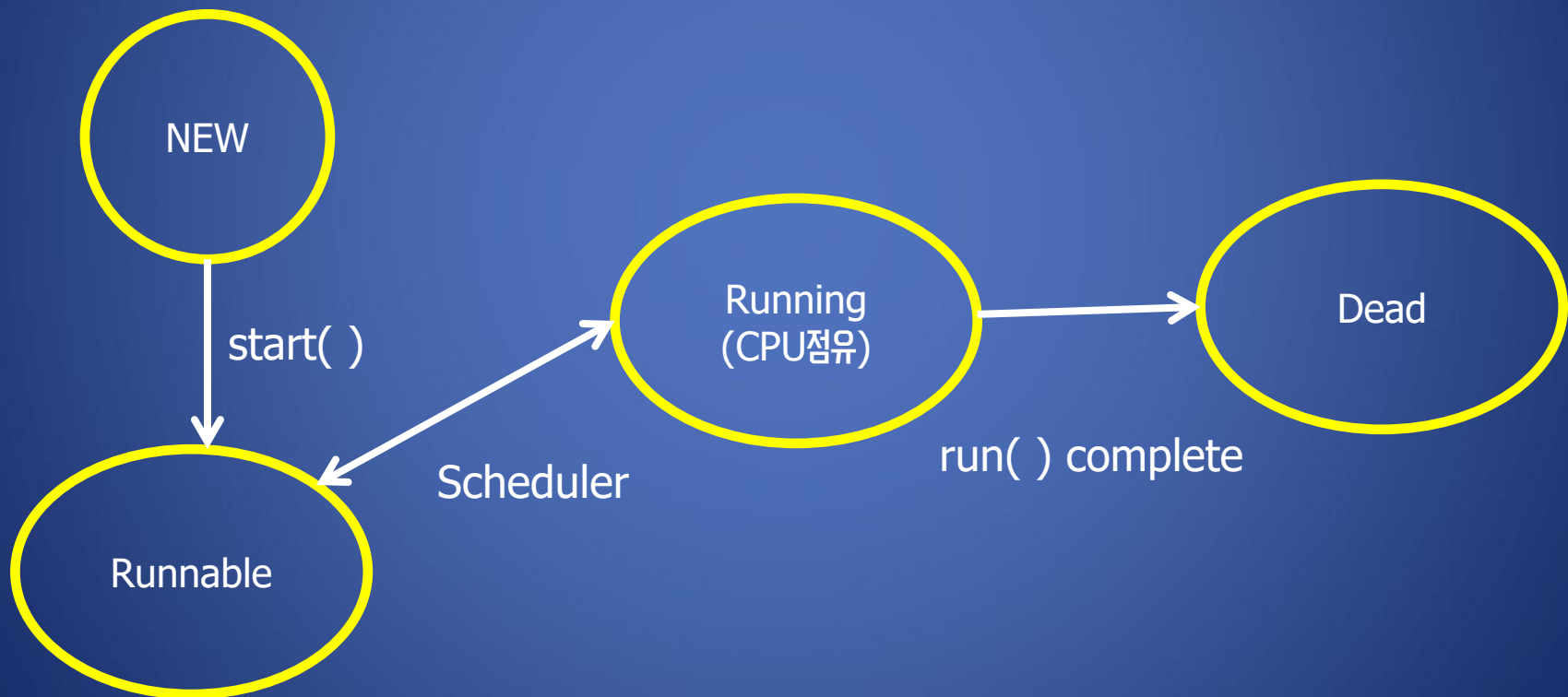
- Java에서는 이런 백그라운드 서비스를 위해 daemon Thread라는 개념을 도입.
- Daemon thread는 앞에서 말한
“프로그램 내부의 모든 thread가 종료되지 않으면 JVM은 종료되지 않는다.”라는 말의 예외.
- 예제 코드 작성

- Daemon Thread

- Daemon thread를 확인하는 방법은 isDaemon() method로 확인 가능.
- 만약, main thread가 생성해서 실행시킨 thread가 종료될 때 까지, main thread가 기다려야 하는 상황이라면?
- join() method 활용 (InterruptedException을 발생시킨다는 것에 주의)
- 예제 코드 작성

Thread

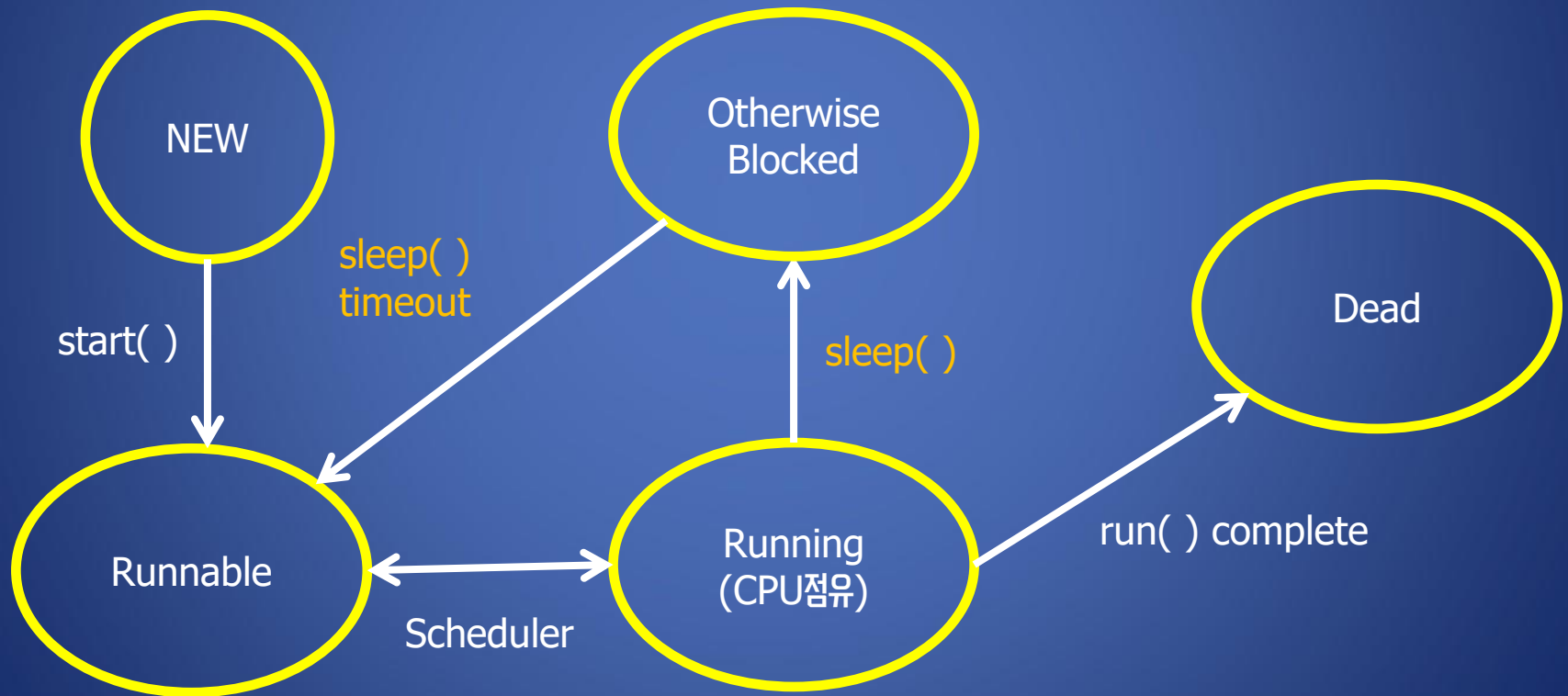
- Java Thread States



- sleep() method
 - CPU점유를 중지하고 특정시간 동안 아무것도 안하고 Thread가 자고 있도록 하는 method

Thread

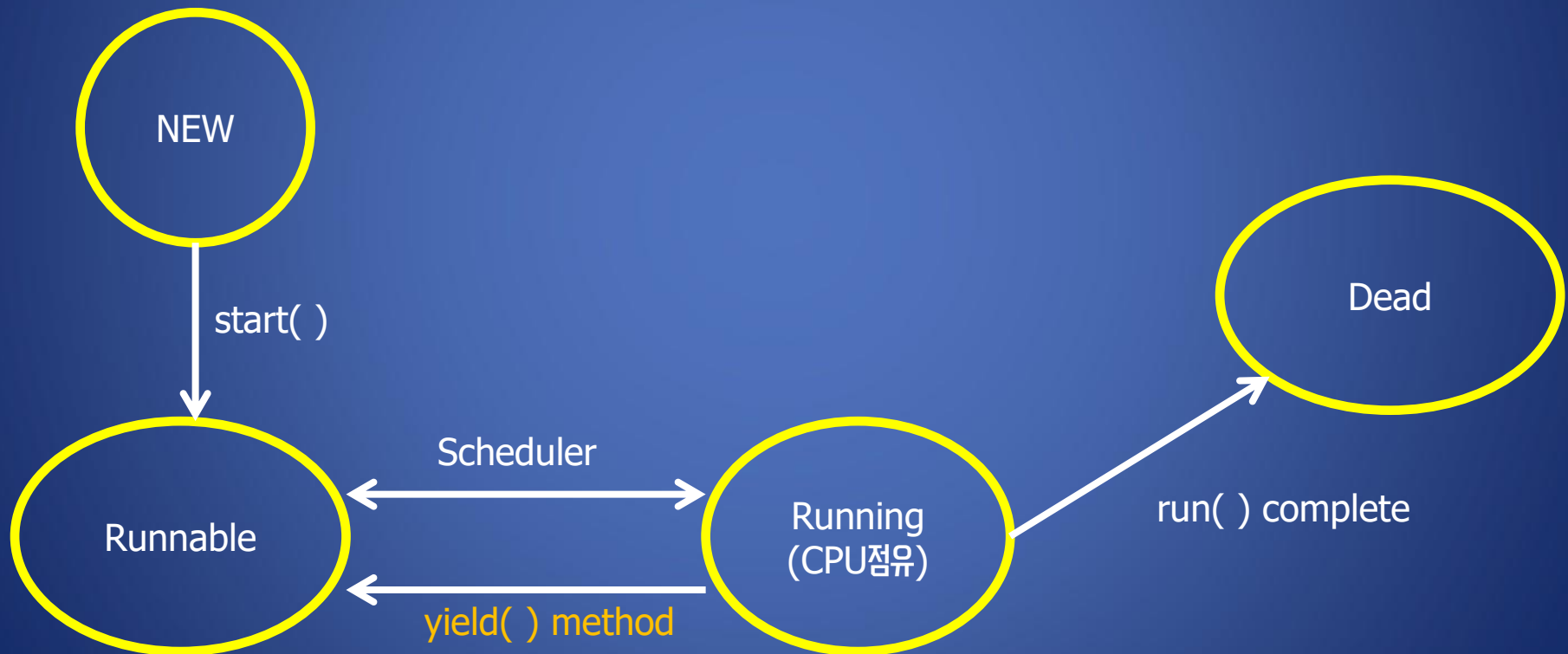
- Java Thread States



- yield() method
 - 강제로 다른 Thread에게 실행상태를 양보하고 자신은 대기상태로 바꾸는 method

Thread

- Java Thread States



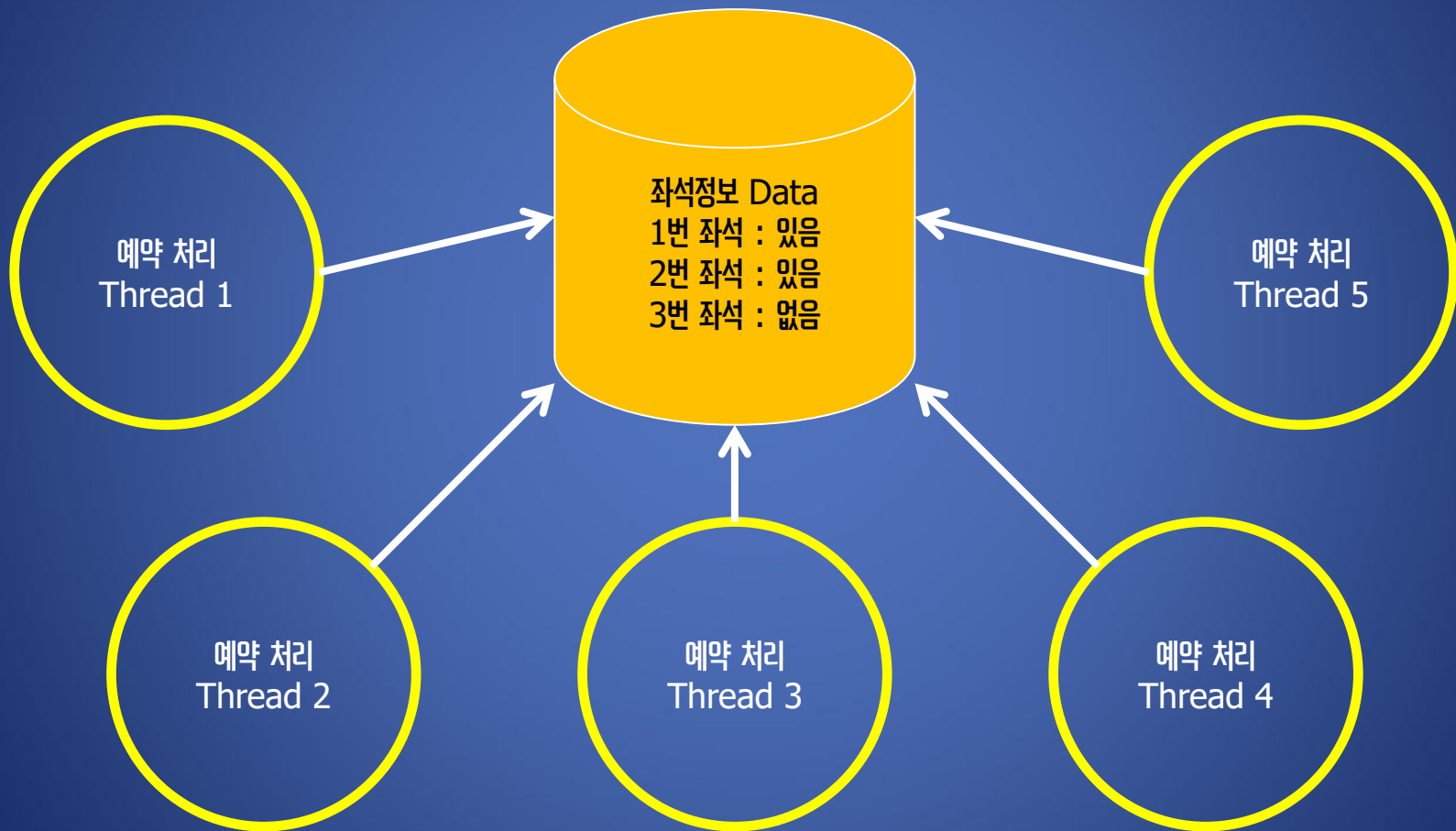
- setPriority() method

- 원칙적으로 각 Thread는 공평하게 CPU를 점유한다.
따라서 Java에서 Thread의 실행순서를 결정하는 것은 매우 어렵다.
- 중요한 Thread인 경우 보다 많은 Running상태로 옮길 수 있는 기회를 주어야 하는데 이때 사용하는 method.
- 우선순위의 값은 1 ~ 10의 값을 가지며 기본적으로 생성한 thread는 5의 값을 가진다.

- 동기화(Synchronization)

- 하나의 자원을 여러 Thread가 사용하려 할 때,
한 시점에 하나의 Thread만이 사용할 수 있도록 하는 것을 의미.
- 예매시스템에 대한 간략한 예.

Thread



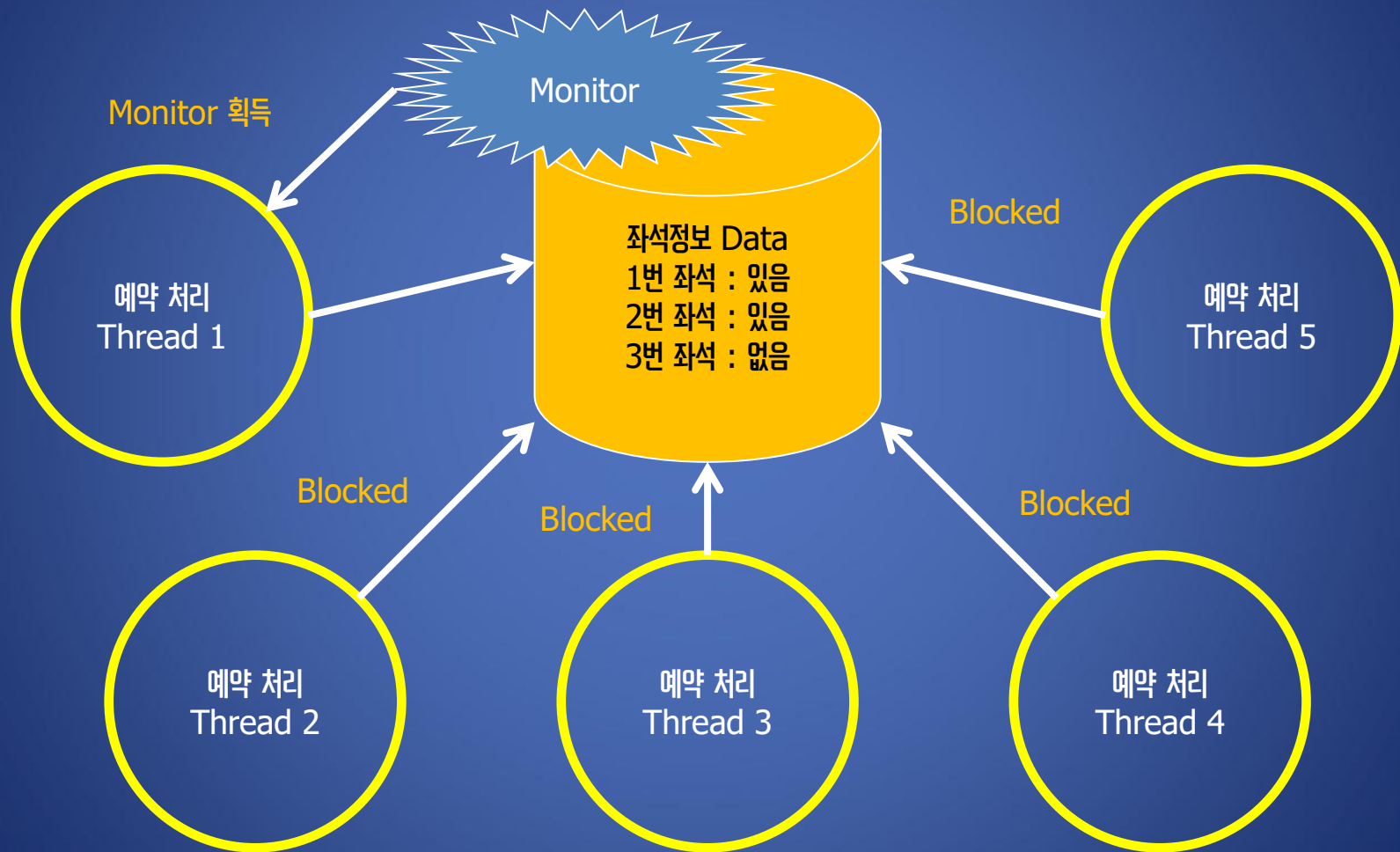
- Monitor

- instance에 Thread가 접근하는 것을 제어하기 위해서 Monitor라는 것을 제공.
- 모든 instance는 각자의 Monitor를 가지고 있다.
- 일단 Monitor를 한 Thread가 가지고 있으면 다른 Thread는 해당 instance의 동기화 method를 실행 시킬 수 없다.

- Monitor

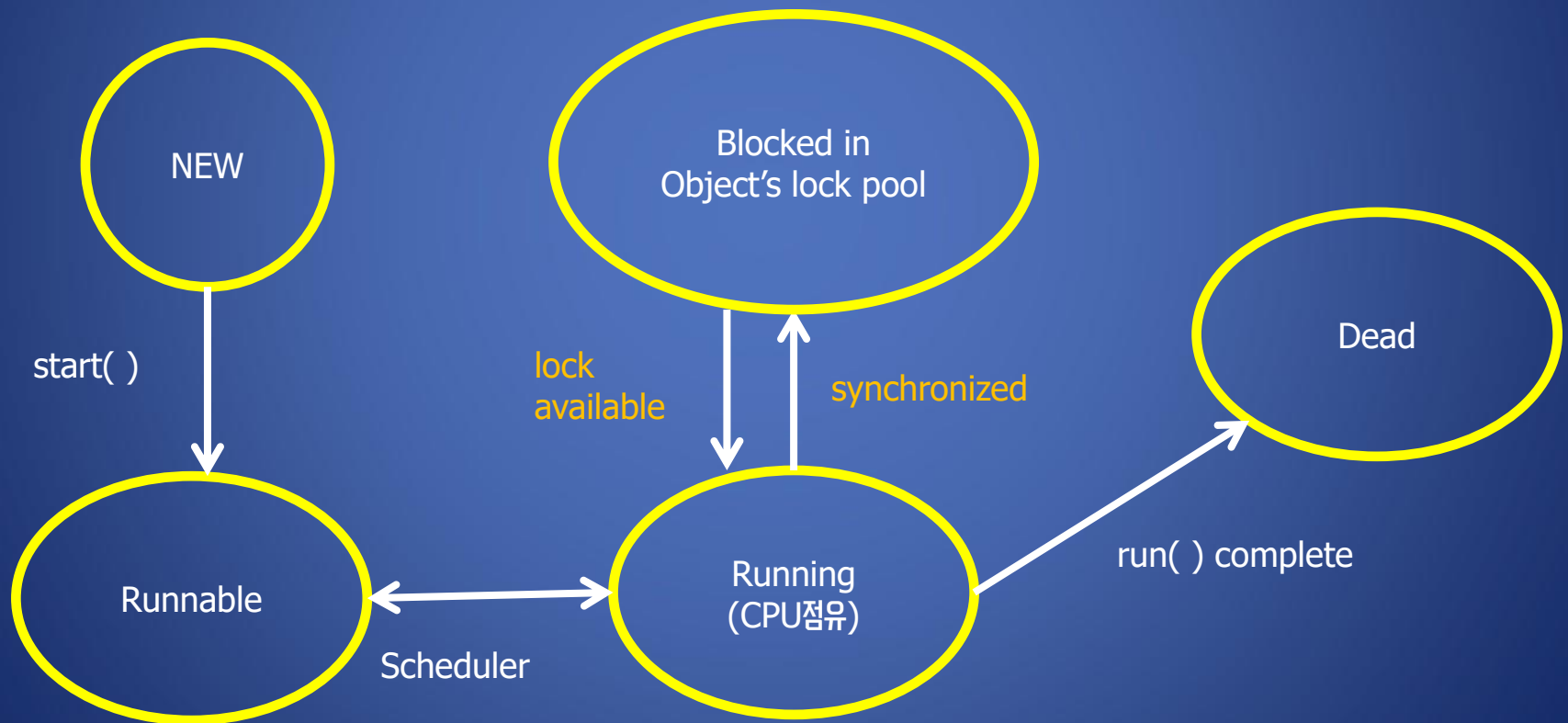
- 그러면 instance의 Monitor를 획득하려면 어떻게 해야 하는가?
- Java에서는 **synchronized keyword**를 제공.

Thread



Thread

- Java Thread States



- wait() , notify() method
 - 여러 Thread가 동시에 하나의 자원을 공유할 때 이 Thread간의 이벤트를 이용한 통신방식이 필요할 경우 사용된다.

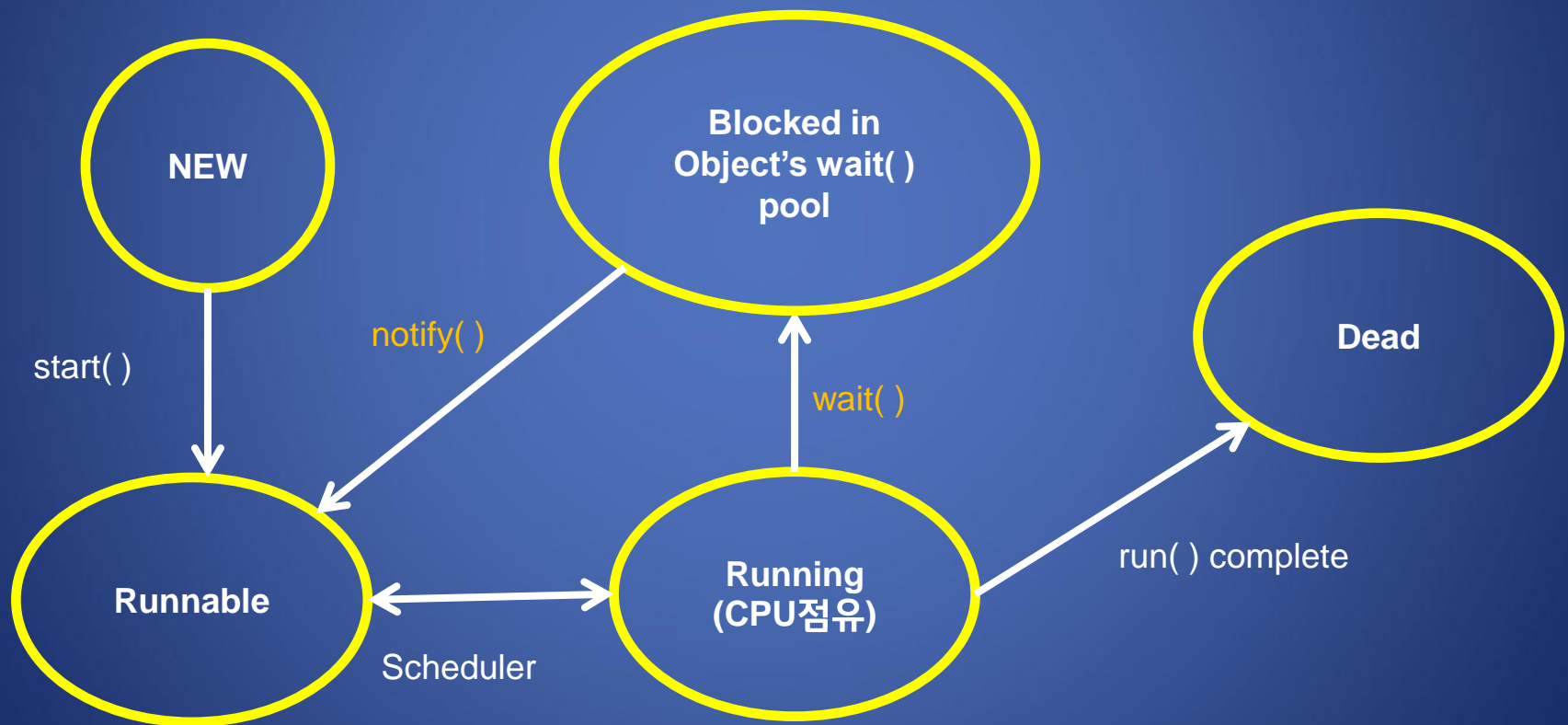
- wait() method

- Monitor를 가지고 있는 Thread가 잠시 정지하고자 하는 경우에 호출.
- 즉, Monitor를 가지고 있는 Thread를 wait상태로 만들고 Monitor를 해제하는 역할.
- 결국 Monitor를 얻기 기다리면서 waiting상태에 있던 다른 Thread들이 동기화 method를 실행시킬 수 있게 해준다.

- notify() , notifyAll() method
 - wait()을 호출하며 waiting상태로 들어간 Thread는 다른 Thread가 notify()나 notifyAll()을 호출함으로 다시 Ready상태로 들어가게 된다.
 - 주의해야 할 점은 해당 method들은 모두 동기화 method, 혹은 동기화 block내에서만 호출이 가능하다는 것.

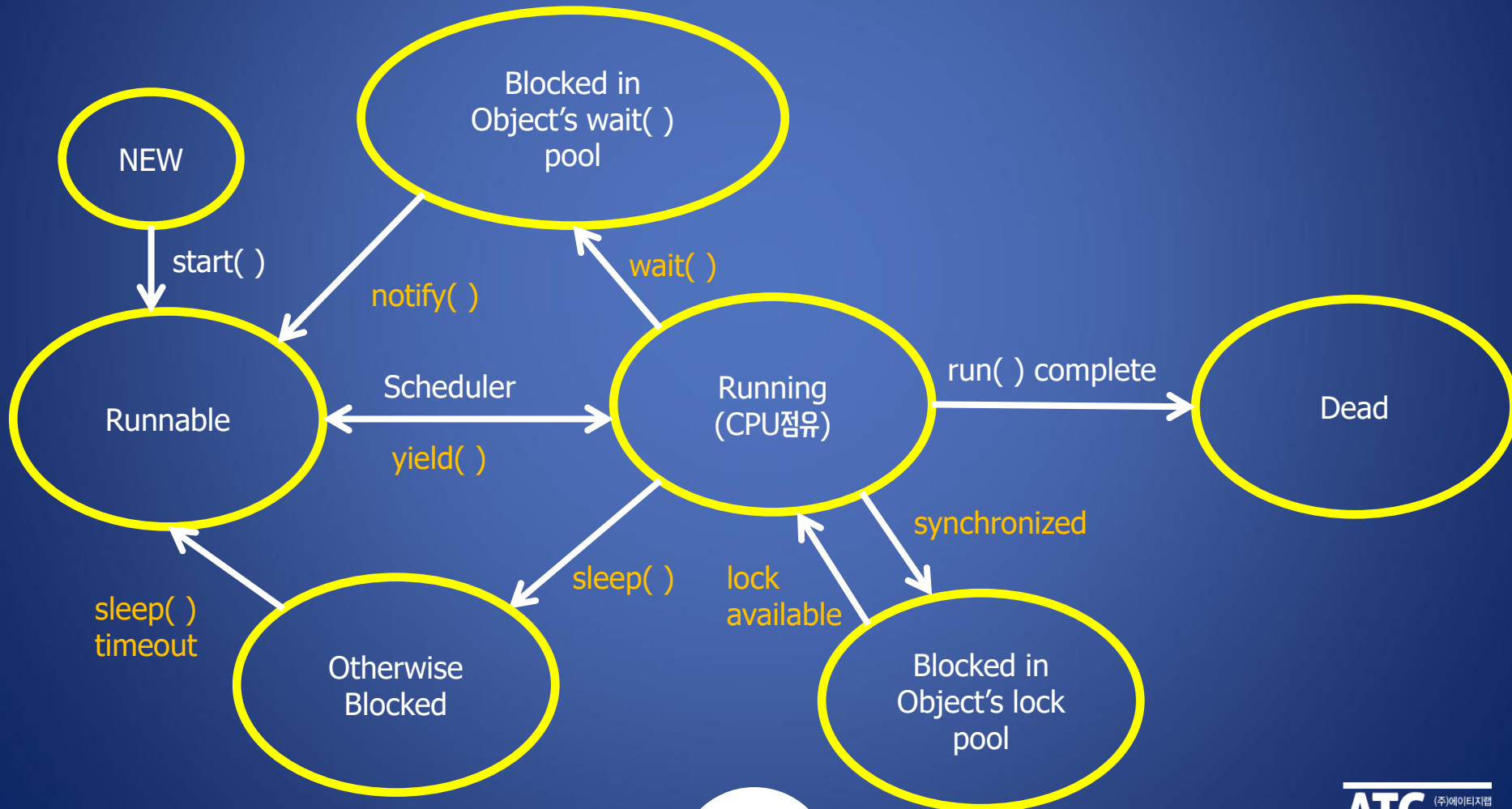
Thread

- Java Thread States



Thread

- Java Thread States



- Singleton Pattern

- 일반적으로 프로그램을 실행할 때 많은 instance가 생성.
- 그러나 class의 instance가 단 1개만 필요한 경우도 있다.
- 물론, new keyword를 이용한 객체생성 관련 코드를 주의해서 작성하면 해결할 수 있지만 ...

이 문제를 프로그램적으로 보증하고 싶을 때 사용

- Singleton Pattern

```
public class Singleton {  
  
    private static Singleton singleton = new Singleton();  
  
    private Singleton() {  
        System.out.println("Singleton 객체 생성");  
    }  
  
    public static Singleton getInstance() {  
        return singleton;  
    }  
}
```


- Producer Consumer Pattern

- Multi Thread를 매기할 때 가장 일반적인 예제로 사용되는 pattern
- Producer와 Consumer가 공통적으로 사용하는 Queue를 프린터 쪽에서 가지고 있다.
(FIFO 구조)

- Producer Consumer Pattern

- Producer는 데이터를 넣고 싶을 때 Queue에 요청을 보낸다. (method 호출)
- Queue는 이런 요청이 들어올 때마다 순서대로 Queue에 넣어준다.
- 실제적인 출력 작업을 하기 위해 Consumer는 주기적으로 Queue를 확인한다.
- 만약 Queue에 데이터가 존재한다면 Queue에 들어온 데이터 하나만 꺼내서 출력한다.
- 반대로 Queue에 저장된 데이터가 없다면 Consumer는 데이터가 들어올 때까지 대기한다.
- Producer는 1개, Consumer는 3개를 작성해서 실행한다.

- Queue interface 설계

- Queue는 Producer와 Consumer가 공통적으로 사용한다.
- Queue class를 작성하기 전에 Producer와 Consumer가 공통적으로 사용하게 될 method를 정의
- Producer는 put() method를 이용하여 데이터를 Queue에 넣는다
- Consumer는 pop() method를 이용하여 Queue에 저장된 데이터를 꺼낸다.
- 예제 코드 작성

- Queue interface 설계

```
public interface Queue {  
  
    public String getName();  
    public void clear(); // Queue의 내용을 모두 지운다.  
    public void put(Object obj); // Queue에 데이터를 넣는다.  
    public Object pop() throws InterruptedException;  
                        // Queue에서 저장된 데이터를 꺼낸다.  
    public int size(); // Queue의 크기를 알아낸다.  
}
```

- Queue interface 설계

```
public class PrinterQueue implements Queue {  
  
    private static final String NAME = "Printer Queue";  
    private static final Object monitor = new Object();  
  
    private LinkedList jobs = new LinkedList();  
    private static PrinterQueue queue = new PrinterQueue();  
  
    public static PrinterQueue getInstance() {  
        if( queue == null ) {  
            synchronized (PrinterQueue.class) {  
                queue = new PrinterQueue();  
            }  
        }  
        return queue;  
    }  
}
```

Thread

- Queue interface 설계

```
@Override
public String getName() {
    return NAME;
}

@Override
public void clear() {
    synchronized (monitor) {
        jobs.clear();
    }
}

@Override
public void put(Object obj) {
    synchronized (monitor) {
        jobs.addLast(obj);
        monitor.notify();
    }
}
```

```
@Override
public Object pop() throws InterruptedException {
    Object o = null;
    synchronized (monitor) {
        if(jobs.isEmpty()) {
            monitor.wait();
        }
        o = jobs.removeFirst();
    }
    return o;
}

@Override
public int size() {
    // TODO Auto-generated method stub
    return jobs.size();
}
```

Thread

- Producer

```
public class Producer implements Runnable {  
  
    private PrinterQueue queue = null;  
    public Producer(PrinterQueue queue) {  
        super();  
        this.queue = queue;  
    }  
    @Override  
    public void run() {  
        System.out.println("Start Producer!!");  
        try {  
            int i = 0;  
            while(!Thread.currentThread().isInterrupted()) {  
                queue.put(i++);  
            }  
        } catch (Exception e) {  
            System.out.println(e);  
        } finally {  
            System.out.println("End Producer!!");  
        }  
    }  
}
```

- Consumer

```
public class Consumer implements Runnable {  
  
    private PrinterQueue queue;  
    private String name = null;  
  
    public Consumer(PrinterQueue queue, String name) {  
        super();  
        this.queue = queue;  
        this.name = "Consumer-" + name;  
    }  
}
```


Thread

- Consumer

```
@Override
public void run() {
    // TODO Auto-generated method stub
    System.out.println("Start " + name);
    try {
        while(!Thread.currentThread().isInterrupted()) {
            System.out.println(name + " : " + queue.pop());
        }
    } catch( Exception e ) {
        System.out.println(e);
    } finally {
        System.out.println("End " + name);
    }
}
```


Thread

- ThreadTest

```
public class ThreadTest {  
  
    public static void main(String[] args) throws Exception {  
  
        PrinterQueue queue = PrinterQueue.getInstance();  
        Thread con1 = new Thread(new Consumer(queue, "1"));  
        Thread con2 = new Thread(new Consumer(queue, "2"));  
        Thread con3 = new Thread(new Consumer(queue, "3"));  
  
        con1.start();  
        con2.start();  
        con3.start();  
  
        Thread pro = new Thread(new Producer(queue));  
        pro.start();  
  
        Thread.sleep(2);  
        pro.interrupt();  
        Thread.sleep(2);  
        con1.interrupt();  
        con2.interrupt();  
        con3.interrupt();  
    }  
}
```

- Java IO ?
 - 자바 입출력 (키보드, 모니터, 네트워크, 파일, etc)
 - Java IO와 관련된 class는 java.io package에 존재
 - Java IO는 stream instance로 처리하며 instance를 만들기 위한 interface와 class가 java.io package에 존재.

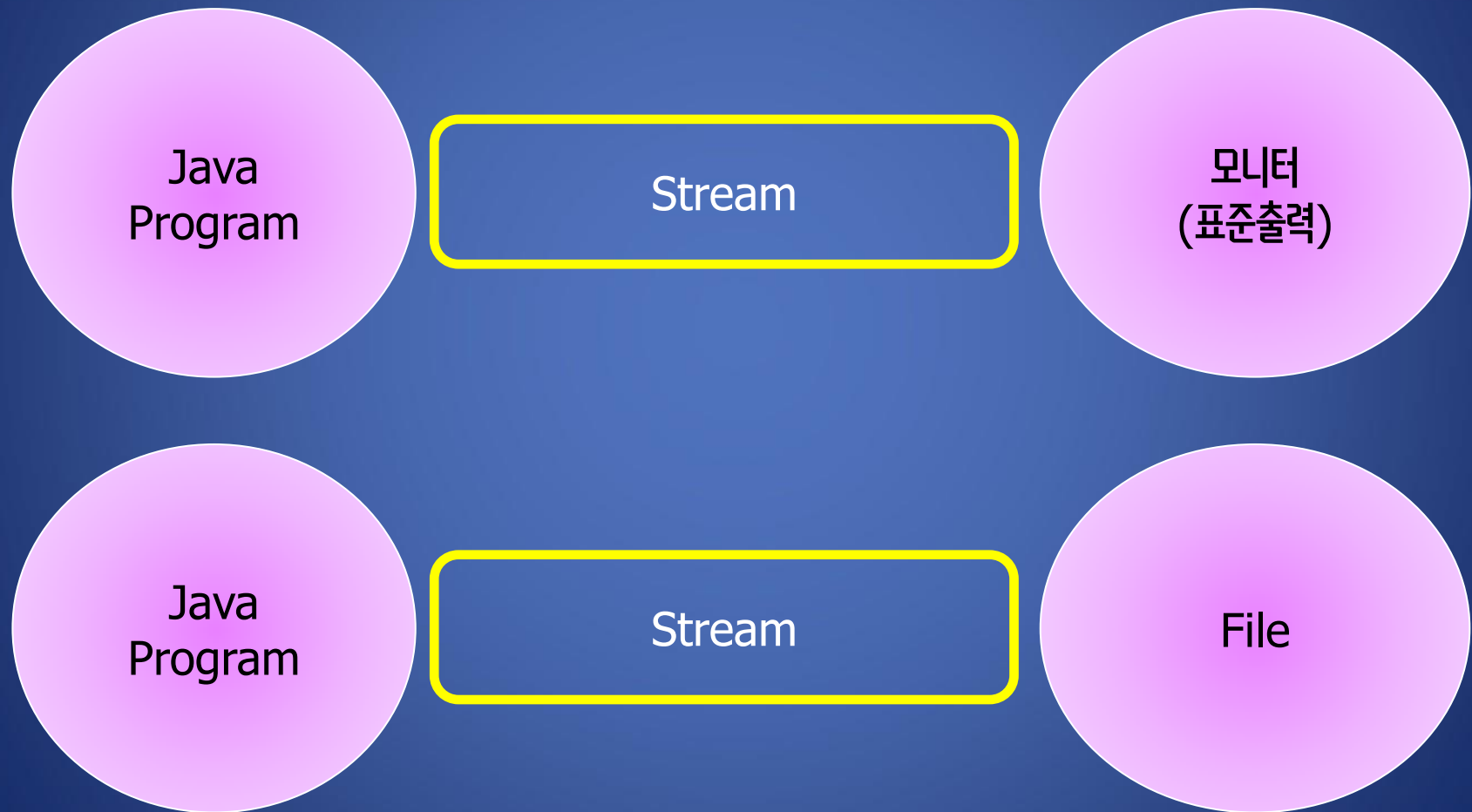
- Java IO에서 가장 특수한 객체
 - System class의 in, out (표준입력, 표준출력)
 - 표준입력 : 보통 keyword의 입력
 - 표준출력 : 화면 출력
 - System.in => java.io.InputStream
 - System.out => java.io.PrintStream



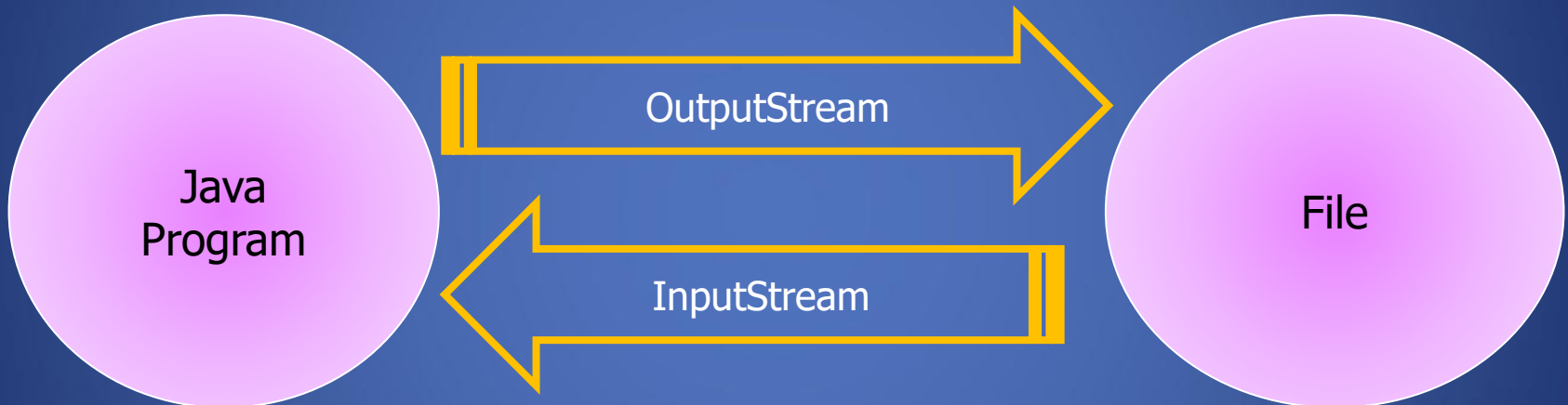
Stream
instance

Stream이란 Java에서 특정장치로부터 data를 읽거나 기록할 때 사용하는 중간 매체.
instance로 사용된다.

Java IO



- Java Stream의 특성



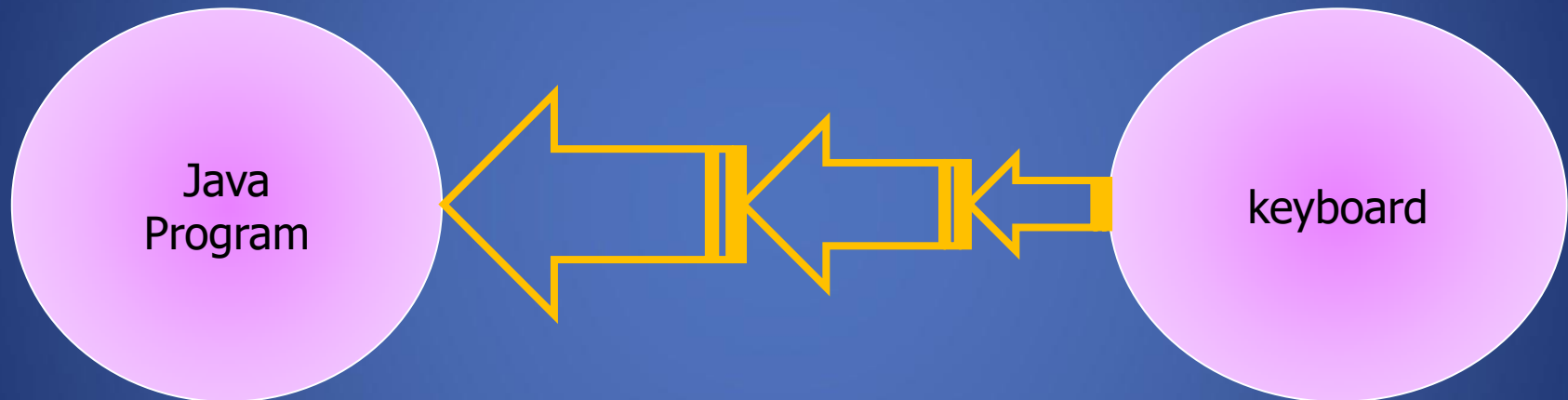
Stream은 단방향 이며 양방향으로 사용될 수 없다.
Stream생성시 Stream의 종류가 구별된다.

- Java Stream의 특성



Stream은 먼저 넣은 데이터가 먼저 나오는 구조.
특정 위치의 Data를 무작위로 읽거나 쓰는 건 원칙적으로 금지.

- Java Stream의 특성



여러 Stream을 결합하여 사용하기 편리한 Stream을 생성해서 사용가능.

- Java Stream의 구분

- InputStream vs. OutputStream
- Byte Stream vs. 문자 Stream (Reader , Writer)
- 하위 stream일 수록 제공하는 method가 하위 level method.
- 따라서, 프로그램을 편하게 작성하기 위해서는 Reader나 Writer 계열의 Stream을 생성해야 하며 Stream의 결합으로 생성할 수 있다.
- 키보드로부터 1줄을 읽어서 출력하는 코드

- Java Stream의 구분

```
public class LineReader {  
  
    public static void main(String[] args) {  
  
        System.out.println("입력하세요.");  
        System.out.print("==> ");  
  
        InputStreamReader isr = new InputStreamReader(System.in);  
        BufferedReader br = new BufferedReader(isr);  
  
        String input = null;  
        try {  
            input = br.readLine();  
        } catch( Exception e ) {  
            System.out.println(e);  
        }  
  
        System.out.println("입력된 문자열은 : " + input);  
    }  
}
```

- File class

- File의 정보를 알아내거나, 파일을 삭제하거나 이름을 변경할 때 사용.
- 디렉토리(폴더) 역시 file로써 취급된다.
- **주요 method**
 - delete() : 파일 삭제 (디렉토리일 경우 빈 디렉토리일 경우만 삭제)
 - exists() : 파일이 존재하는지에 대한 여부
 - deleteOnExit() : JVM이 종료될 때 파일 삭제
 - getName() : 파일이나 디렉토리명 반환
 - getPath() : 파일의 경로를 문자열 형태로 반환

- File class

```
public class FileDelete {  
  
    public static void main(String args[]) {  
  
        File f = new File("D:/myFile.txt");  
  
        if(f.exists()) {  
            boolean result = f.delete();  
            if(result) {  
                System.out.println("파일 삭제 성공!");  
            } else {  
                System.out.println("파일 삭제 실패!");  
            }  
        } else {  
            System.out.println("파일이 존재하지 않아요!");  
        }  
    }  
}
```

- File class

- 프로그램을 작성하다 보면, 임시 파일을 생성할 필요가 있다.
- 외부 파일을 이용해서 정렬 작업을 해야 할 경우
- 문제는 임시파일명을 어떻게 정할 까 하는 점.
(동일한 이름이 있을 경우 파일이 생성되지 않는다.)
- File class는 임시파일을 생성하기 위한 createTempFile() static method 제공
- 파일명의 prefix와 suffix를 지정해 주면 중간내용은 유일한 이름으로 생성

- File class

```
public class CreateTempFile {  
  
    public static void main(String[] args) {  
  
        try {  
            File tmpFolder = new File("D:/");  
            File f = File.createTempFile("moon_", ".dat", tmpFolder);  
            System.out.println("15초 동안 잠시 재웁니다.");  
            try {  
                Thread.sleep(15000);  
            } catch( InterruptedException e1 ) {  
                System.out.println(e1);  
            }  
            f.deleteOnExit();  
        } catch( IOException e ) {  
            System.out.println(e);  
        }  
        System.out.println("프로그램 종료!!");  
    }  
}
```

- 문자 단위 IO class

- 바이트 단위가 아닌 문자 단위로 입출력.
- 입력 class인 경우 Reader로 끝나고, 출력 class인 경우 Writer로 끝남.
- FileReader와 FileWriter를 이용한 file의 복사

- 문자 단위 IO class

```
public static void main(String[] args) {  
  
    FileReader fr = null;  
    FileWriter fw = null;  
  
    try {  
        fr = new FileReader("D:/original.txt");  
        fw = new FileWriter("D:/target.txt");  
  
        char[] buffer = new char[512];  
        int readCount = 0;  
  
        while((readCount = fr.read(buffer)) != -1) {  
            fw.write(buffer,0,readCount);  
        }  
    } catch( Exception e ) {  
        System.out.println(e);  
    } finally {  
        System.out.println("파일복사를 완료했습니다.");  
        try {  
            fr.close();  
            fw.close();  
        } catch( Exception e1 ) {  
            System.out.println(e1);  
        }  
    }  
}
```

- 문자 단위 IO class

- BufferedReader와 BufferedWriter를 이용하여 속도를 높여보자.
- Buffer를 이용한 입출력 시에 병목현상을 제거
- 특히 BufferedReader같은 경우 readLine() method가 있어 프로그램 처리에도 유리.
- 단, 버퍼를 사용할 경우 반드시 flush()나 close()를 호출해야만 한다.
그렇지 않으면 버퍼의 내용을 잃어버릴 수 있다.

- 문자 단위 IO class

```
public static void main(String[] args) {
    FileReader fr = null;
    FileWriter fw = null;
    BufferedReader br = null;
    BufferedWriter bw = null;
    try {
        fr = new FileReader("D:/original.txt");
        fw = new FileWriter("D:/target.txt");
        br = new BufferedReader(fr);
        bw = new BufferedWriter(fw);

        char[] buffer = new char[512];
        int readCount = 0;

        while((readCount = br.read(buffer)) != -1) {
            bw.write(buffer,0,readCount);
        }
    } catch( Exception e ) {
        System.out.println(e);
    } finally {
        System.out.println("파일복사를 완료했습니다.");
        try {
            br.close();
            bw.close();
        } catch( Exception e1 ) {
            System.out.println(e1);
        }
    }
}
```

- 문자 단위 IO class
 - 출력 시에 유용한 class : `PrintWriter`
 - `print()`, `println()`과 같은 유용한 method를 가지고 있다.
 - API Reference를 참조하여 다음장의 예제를 작성

- 키보드로부터 한 줄씩 입력 받아 파일에 저장하는 예제 작성
- 키보드로부터 입력을 종료할 때는 ctrl-z 를 입력

- JDBC를 이용하여 제공된 library database에서
키워드로 검색된 책의 정보를 얻어와 특정 file에 저장하는 프로그램을
작성해 보자.
- 별도의 프로그램을 작성하여 해당 file로부터 데이터를 읽어 들여
책 제목으로 오름차순 정렬하여 두 번째 파일에 저장해보자.

- Object Stream

- 지금까지는 간단한 데이터를 읽고 쓰는데 초점을 맞추었는데 복잡한 데이터를 Stream을 통해서 읽고 쓰려면 어떻게 해야 하는가?
- 즉, 객체를 Stream을 통해서 보내고 받기 위해서 어떻게 해야 하는가?
- 예로 알아보자. (File에 HashMap을 저장 한 후 다시 읽어 들여보자)

- Object Stream

- Object Stream은 메모리 내의 복잡한 내용을 쓰고 읽기에 아주 편리한 기능을 제공
- 객체를 전송하기 위해서는 데이터를 특정 통신 채널로 보낼 수 있는 형태로 변환시킨다.
(마샬링 – Marshaling)
- 변환된 데이터를 전송한다.
- 변환된 데이터를 읽어 들여 원래 형태로 변환시킨다.
(언마샬링 – Unmarshaling)

- Object Stream

- 만약 사용자 정의 class의 객체를 Object Stream을 이용해서 사용하려면 해당 class는 반드시 **Serializable interface**를 구현해야 한다.
- 당연한 말이지만 객체 직렬화가 불가능한 객체는 전송될 수 없다.
- 예제로 알아보자.

- Object Stream

- 만약 사용자 정의 class의 객체를 Object Stream을 이용해서 사용하려면 해당 class는 반드시 Serializable interface를 구현해야 한다.
- 당연한 말이지만 객체 직렬화가 불가능한 객체는 전송될 수 없다.

Java Network - Terminology

Internet

IP
address

DNS

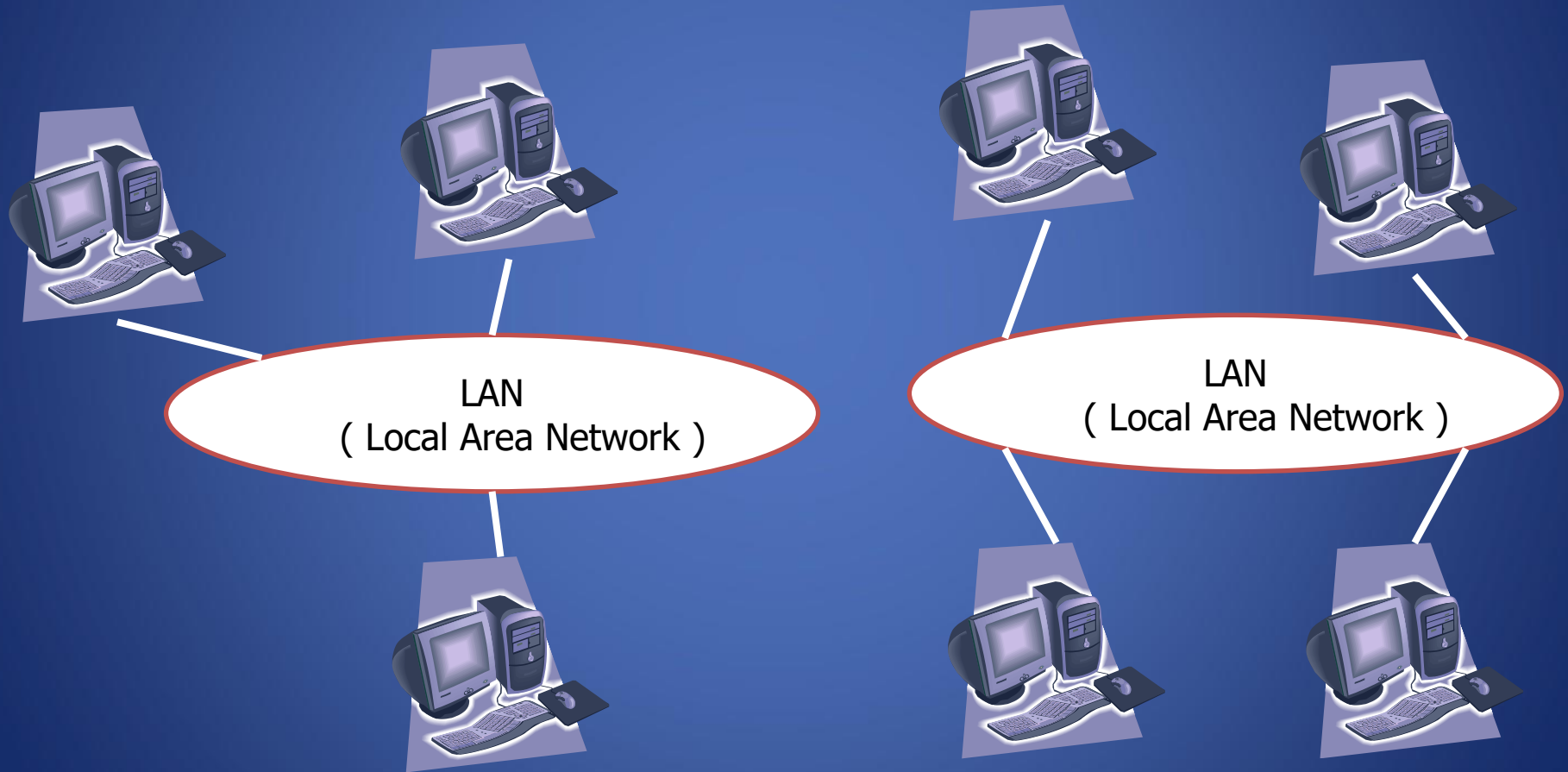
Protocol

Gateway

Router

Port

Java Network - Terminology



Java Network - Terminology

LAN

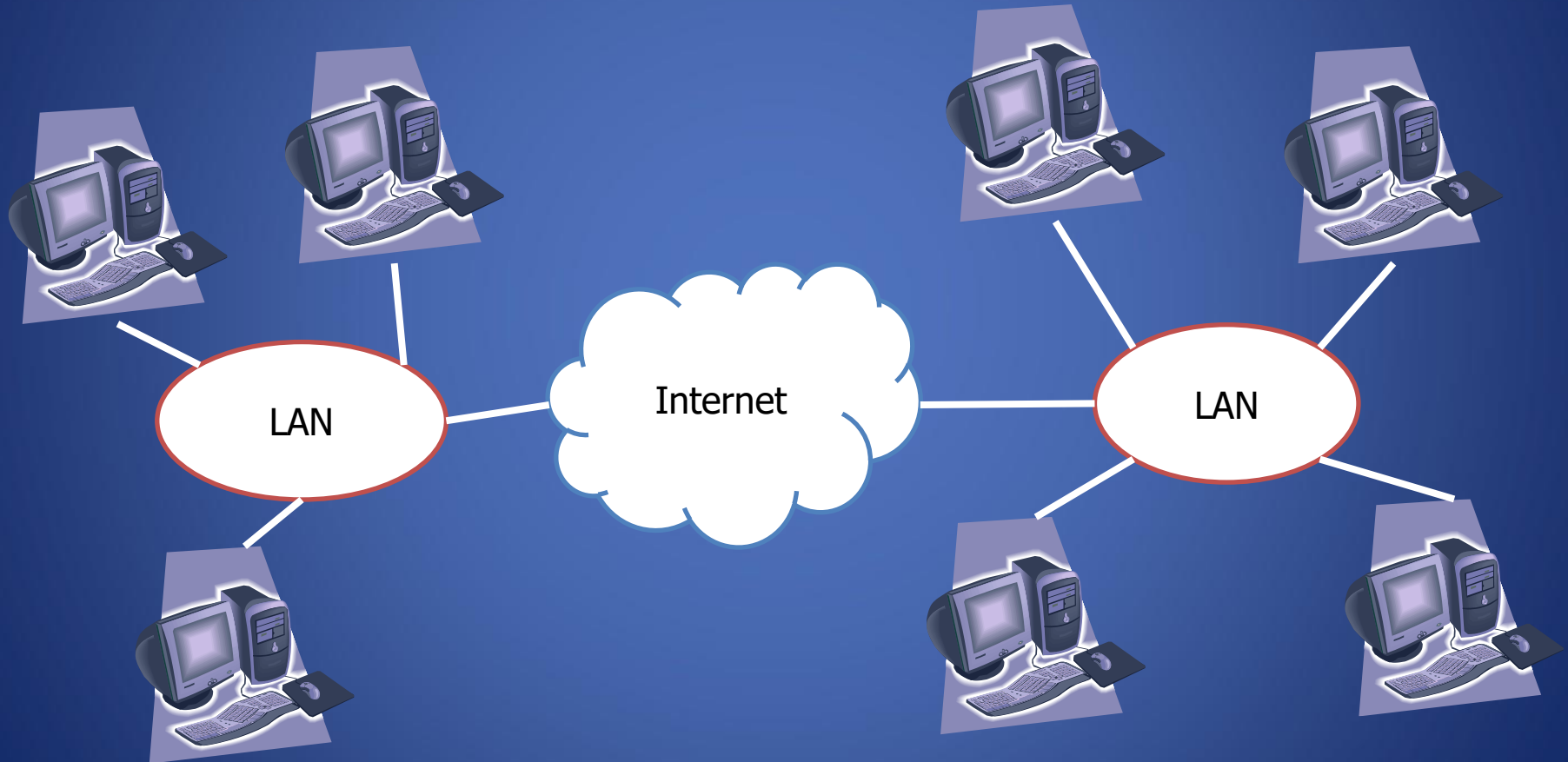
MAN

WAN

Network의 크기에 따른 구분

Internet : Network of Network

Java Network - Terminology



Java Network - Terminology

IP address

컴퓨터를 구분하기 위한 논리적인 주소 (4byte – 32bit)

IPv6 (16byte – 128 bit)

MAC Address (48 bit)

Java Network - Terminology



DNS



Domain Name System

Domain Name Server

Java Network - Terminology

Protocol

통신규약

TCP / IP
ARP
FTP
TELNET
HTTP
etc..

Java Network - Terminology



Gateway

하나의 network에서 다른 network로 들어가는
입구역할을 하는 장치

하나의 network를 다른 network과 연결

왜 필요한가 ?

Protocol의 변환, 컴퓨터 address의 변환,
통신속도의 제어

Java Network - Terminology

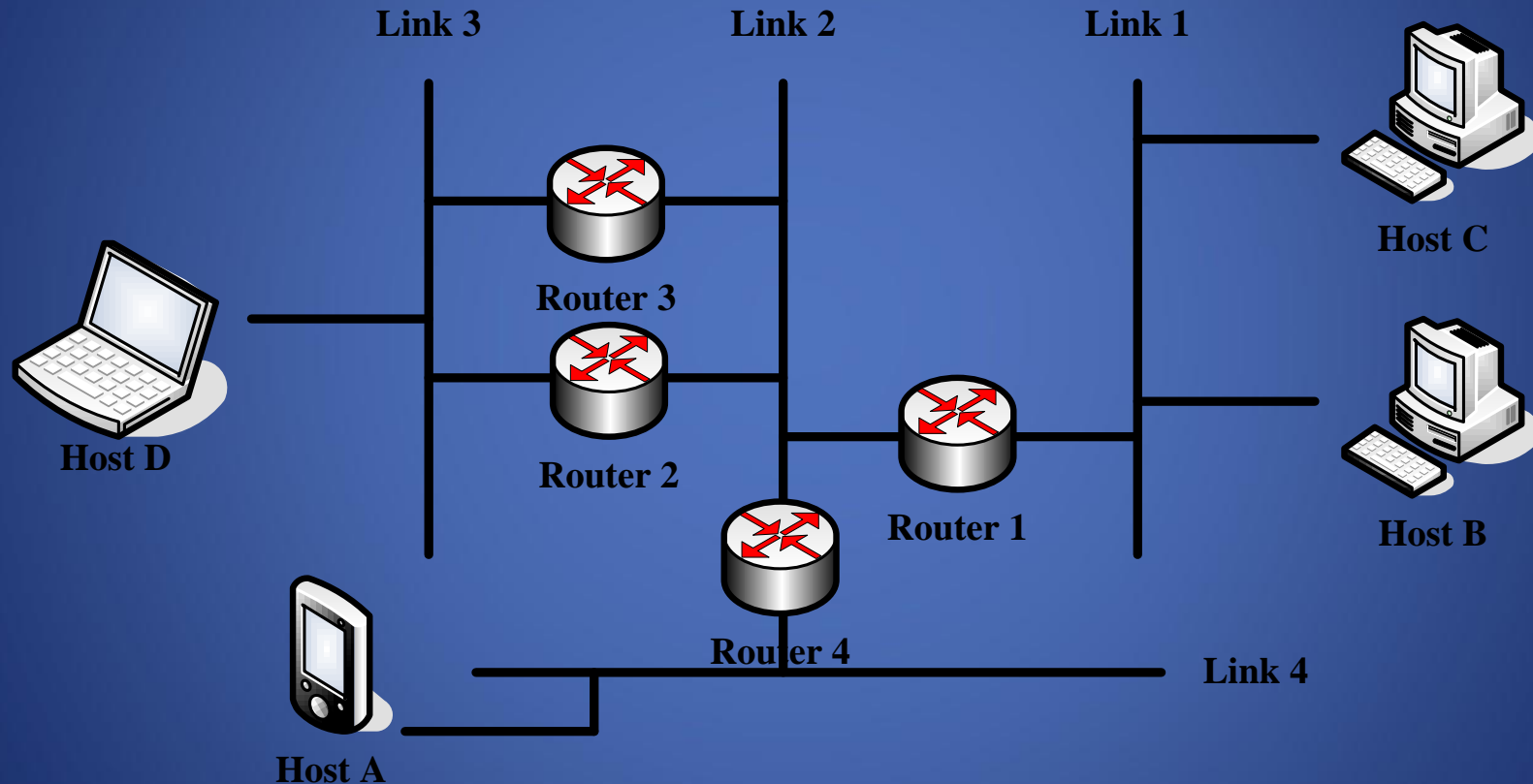


Router

서로 다른 network를 중계해 주는 장치

수신처의 주소를 읽어서 가장 적절한 통신로를 지정하고
전송하는 장치

Java Network - Terminology



Java Network - Terminology



Port

16bit의 숫자

Application을 인지하기 위하여 사용되는 숫자

0~1023 reserved

Java Network - Terminology



Socket

네트워크에서 데이터가 전송되기 위해서는 packet이라는 단위로
쪼개져서 데이터가 전송.

받는 측에서는 이러한 packet을 합쳐서 데이터를 복원하여 사용.

이러한 동작을 해주는 것이 바로
TCP/IP protocol

이런 TCP/IP프로토콜을 이용하기 위해서 사용하는 것이
Socket API(java에서는 Socket class)

Socket이라는 개념은
맨 처음 BSD계열의 UNIX에서 처음 소개

하위의 복잡한 프로토콜과 상관없이 좀 더 쉽고 유연하게 네트워크 프로그램을 작성

Socket은 data를 보내거나 받는 창구역할.

Port를 이용한 Data의 전송을 추상화 시킨 것.

일단 Socket을 이용하여 상대 측과 접속하게 되면, 마치 file system이나 다른 I/O처럼
사용이 가능

Socket 프로그래밍을 하기 위해서는 우리가 작성하는 프로세스가 서버역할을 하는지 클라이언트 역할을 하는지 결정.

클라이언트의 역할을 할 때와 서버의 역할을 할 때 Socket class를 사용하는 방법에 차이가 있다.

Java Network - Terminology



Client Program



Server program

- Socket의 형식

- Socket의 형식에는 3가지가 존재
- SOCK_STREAM, SOCK_DGRAM, SOCK_RAW
- SOCK_RAW는 보안상 지원하지 않는다.
- SOCK_STREAM 형식을 이용하는 통신방식을 TCP (Transfer Control Protocol)
- SOCK_DGRAM 형식을 이용하는 통신방식을 UDP(User Datagram Protocol)

Java Network - TCP

- Java언어가 제공하는 Socket프로그램은 크게 2가지 종류
- TCP & UDP
- TCP의 특징
 - Stream 통신 protocol
 - TCP통신을 하려면 양쪽의 Socket이 연결된 상태여야만 가능
 - 연결지향 protocol
 - 송신한 데이터가 차례대로 목적지의 socket에 전달되는 신뢰성 있는 통신이 가능

Java Network - TCP

- 서버는 클라이언트가 세션을 요청하면, 이에 대한 응답을 하는 프로세스를 의미.
- 서버도 클라이언트와 마찬가지로 Socket class를 이용해야 네트워크 통신을 할 수 있다.
- 그러나, 서버는 클라이언트와 달리 클라이언트의 접속을 대기하고 있어야 한다.

Java Network - TCP

- Java언어에서는 TCP 프로그래밍을 쉽게 할 수 있도록 java.net package에 관련 class들을 제공
- TCP 프로그래밍에서 가장 중요한 class는
 - **java.net.Socket** : 서버와 클라이언트가 통신하기 위해서 필요한 class
 - **java.net.ServerSocket** : 서버에서 클라이언트의 접속을 대기하기 위해서 필요한 class. ServerSocket은 특정 포트에서 대기하고 있다가, 접속 요청이 있으면 이를 새로운 Socket으로 연결하는 역할

- 서버와 클라이언트의 통신 순서 - 1

- 서버 쪽에서는 client의 접속을 기다리기 위해 `ServerSocket` 객체를 생성한 후 `accept()`를 호출
- `accept()` method는 blocking method

```
try {  
    ServerSocket server = new ServerSocket(5000);  
    System.out.println("서버가 접속을 기다립니다.");  
    Socket s = server.accept();  
} catch (Exception e) {  
    System.out.println(e);  
}
```

- 서버와 클라이언트의 통신 순서 - 2

- 클라이언트에서는 Socket객체를 생성하면 서버에 접속할 수 있다.
- 이때 접속한 성공하면 실제로 Socket 객체가 반환

```
Socket s = new Socket("127.0.0.1", 5000);
```


- 서버와 클라이언트의 통신 순서 - 3

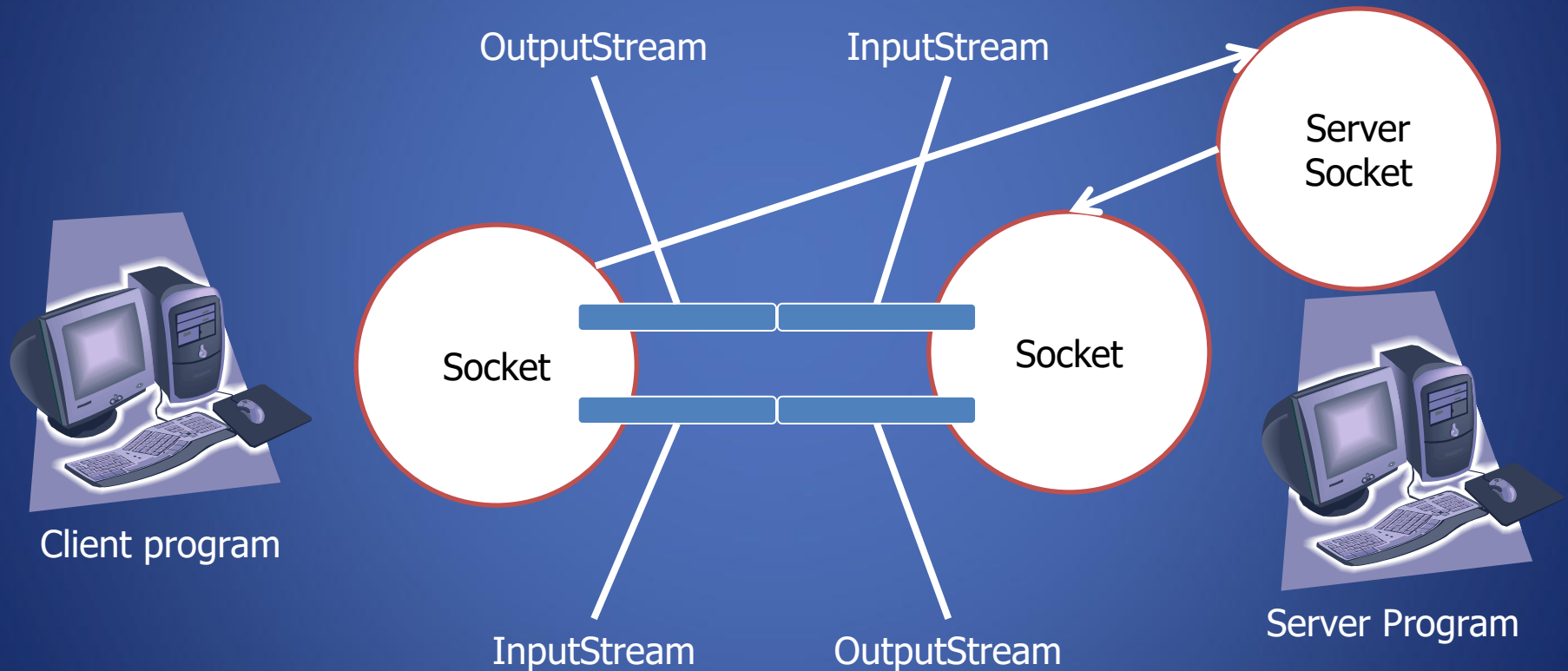
- 데이터 통신을 하기 위해 클라이언트와 서버가 각각 생성한 **socket에 대해서 Stream을 생성한다.**
- 이때, 서버의 OutputStream은 클라이언트의 InputStream과 연결되고 반대로 서버의 InputStream은 클라이언트의 OutputStream과 연결된다.

```
OutputStream out = sock.getOutputStream();  
InputStream in = sock.getInputStream();
```

- 서버와 클라이언트의 통신 순서 - 4

- 접속을 종료할 때는 socket에 대해서 close() method를 호출한다.
- 주의할 점은 socket으로 부터 얻은 Stream이 존재할 때는 해당 IO객체를 먼저 close() 해 주어야 한다.

Java Network - TCP



Java Network - TCP

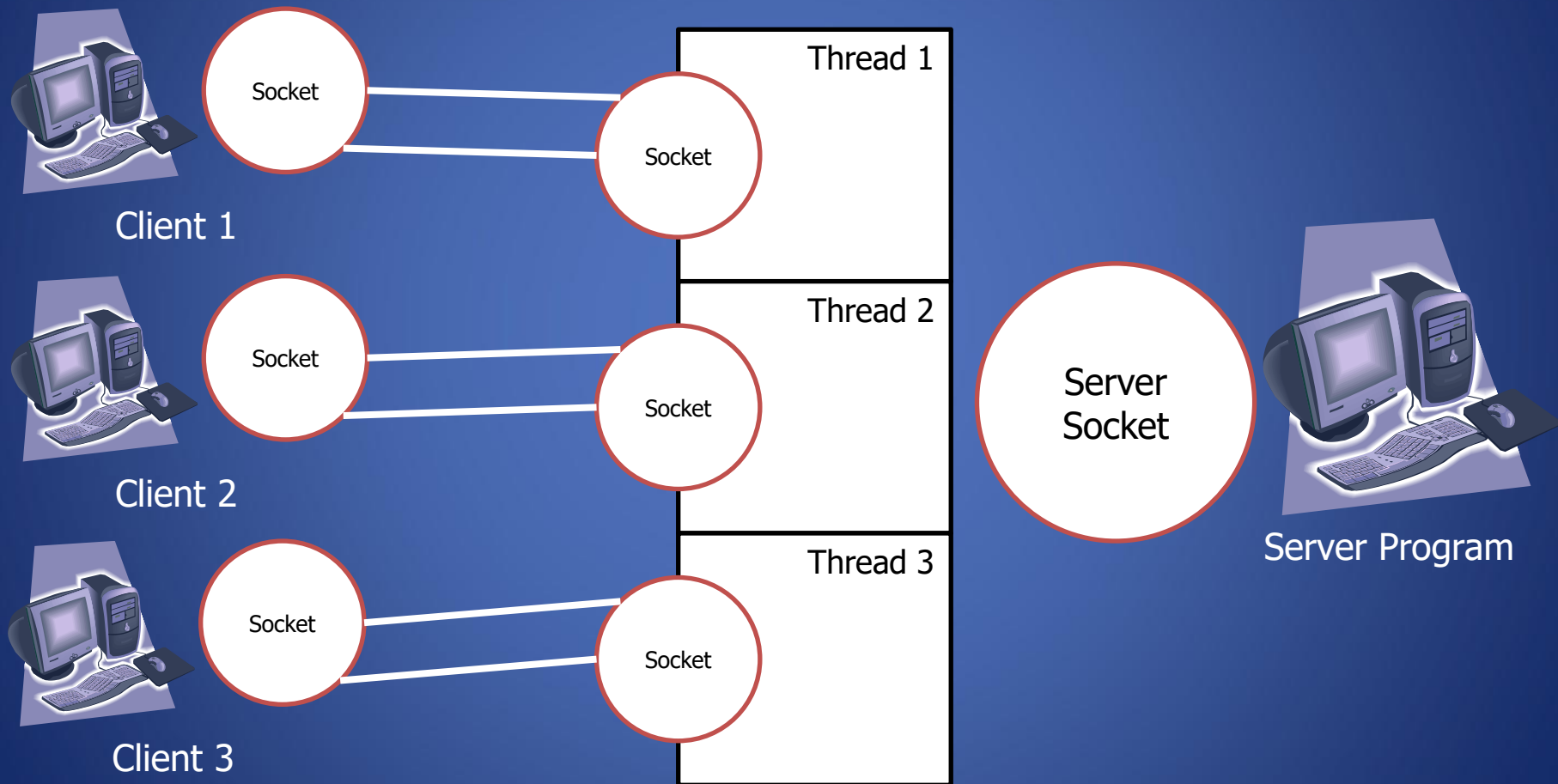
- 접속한 클라이언트에게 날짜를 전송하는 서버프로그램을 작성해 보자.
- 클라이언트가 서버에 접속하면 서버는 현재 시간을 구해서 클라이언트에게 전송하고 서버프로그램을 종료.
- 클라이언트는 서버가 보내준 데이터를 화면에 출력하고 프로그램 종료

Java Network - TCP

- 간단한 Echo 클라이언트/서버 프로그램 작성
- Echo 프로그램이란 클라이언트가 보낸 문자열을 서버가 받아서 다시 클라이언트에게 재 전송해주는 프로그램
- 단, 프로그램은 클라이언트가 “/exit”를 입력하면 종료하는 것으로 작성

- 이전의 code는 1명의 클라이언트에 대해서는 잘 동작하지만 다수의 클라이언트에 대해서는 서비스를 제공하지 못한다.
- 다수의 클라이언트에게 똑같은 Echo 서비스를 제공해 주기 위해서는 어떻게 해야 하는가?
- Multi Thread 이용

Java Network - TCP



Java Network - TCP

- 다수의 클라이언트에게 똑같은 Echo 서비스를 제공하는 프로그램 작성

- 간단한 Web Server 작성

- Web browser는 Web Server에게 HTTP protocol을 이용하여 페이지를 요청하고 그 결과를 화면에 출력
- 간단한 Web Server를 Socket 프로그래밍을 이용해서 직접 구현해 보자.
- HTTP Protocol
- 클라이언트 browser는 server에 request(요청정보)를 보내고 Web Server는 해당 요청을 분석해서 response(응답정보)를 browser에게 전송
- 간단한 형식의 protocol이기 때문에 널리 사용.

- 간단한 Web Server 작성

- 제일 먼저 browser의 요청정보를 출력하는 프로그램을 작성
- 요청정보의 맨 첫 줄에 요청 방식, 요청 파일, 사용한 프로토콜 버전이 명시되는 것을 확인하자.
- 기존의 Echo Server 프로그램을 수정한 후 실행

- 간단한 Web Server 작성

- 요청하는 파일에 대한 정보를 문자열 처리를 통해서 획득
- 해당 HTML파일을 작성한 후 file에 대한 Stream을 통해서 file의 내용을 클라이언트 web browser에 전송
- 만약 MIME type때문에 정상적으로 표현이 되지 않는다면 HTTP protocol의 response header에 대한 정보를 포함해서 정상적으로 표현되게 작성해 보자.

Java Network - TCP

- 방 1개 짜리 채팅프로그램 작성

- 지금까지의 내용을 토대로 chatting 프로그램을 작성해 보자.

- 여러 개의 방이 있는 채팅프로그램 작성

- 이전 문제를 확장하여 여러 개의 방이 있는 채팅 프로그램 작성
- 추가기능을 생각해서 프로그램에 적용해 보자.
- 예) 귓속말 기능, 모든 방에 있는 사람에게 전체 메시지 보내는 기능 등등.

- Java Servlet을 사용한 Web Programming

- 실습으로 진행.

Thanks for listening

