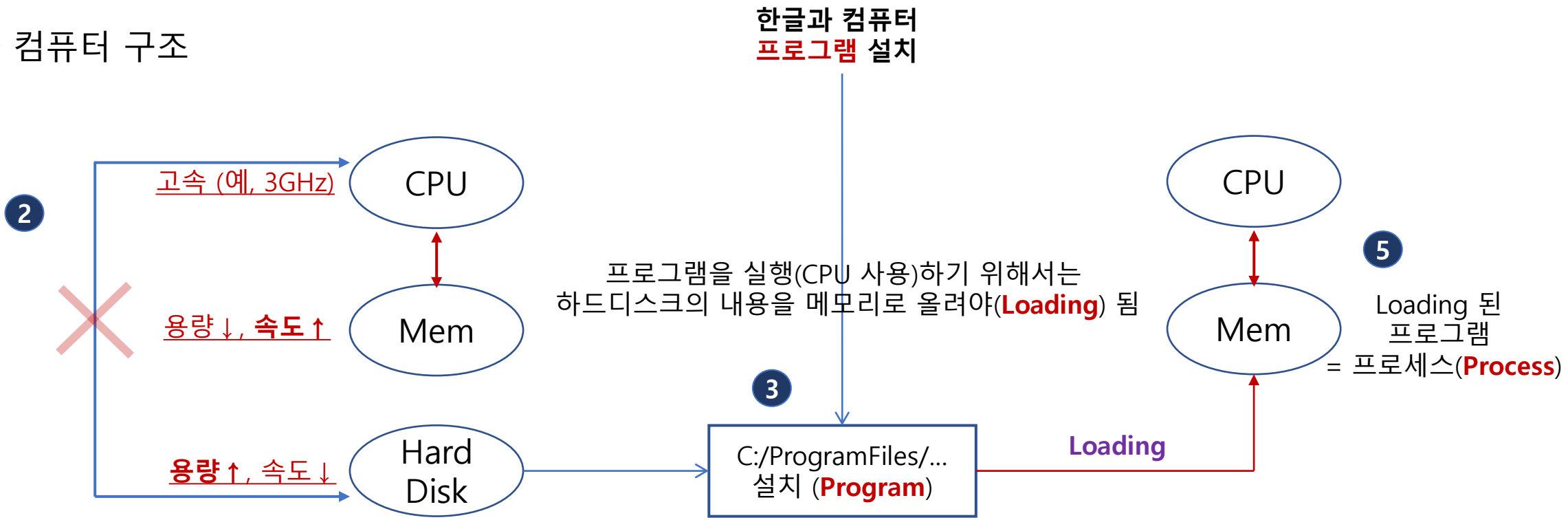
쓰레드(Thread)

Program vs. Process vs. Thread의 개념

1 Program vs. Process vs. Thread의 개념

☞ 컴퓨터 구조



4 **Q. 말이 빠른 사람과 느린 사람이 대화하면 누구 속도에 맞춰질까?**
→ CPU는 Hard Disk와는 직접 이야기 하지 않음

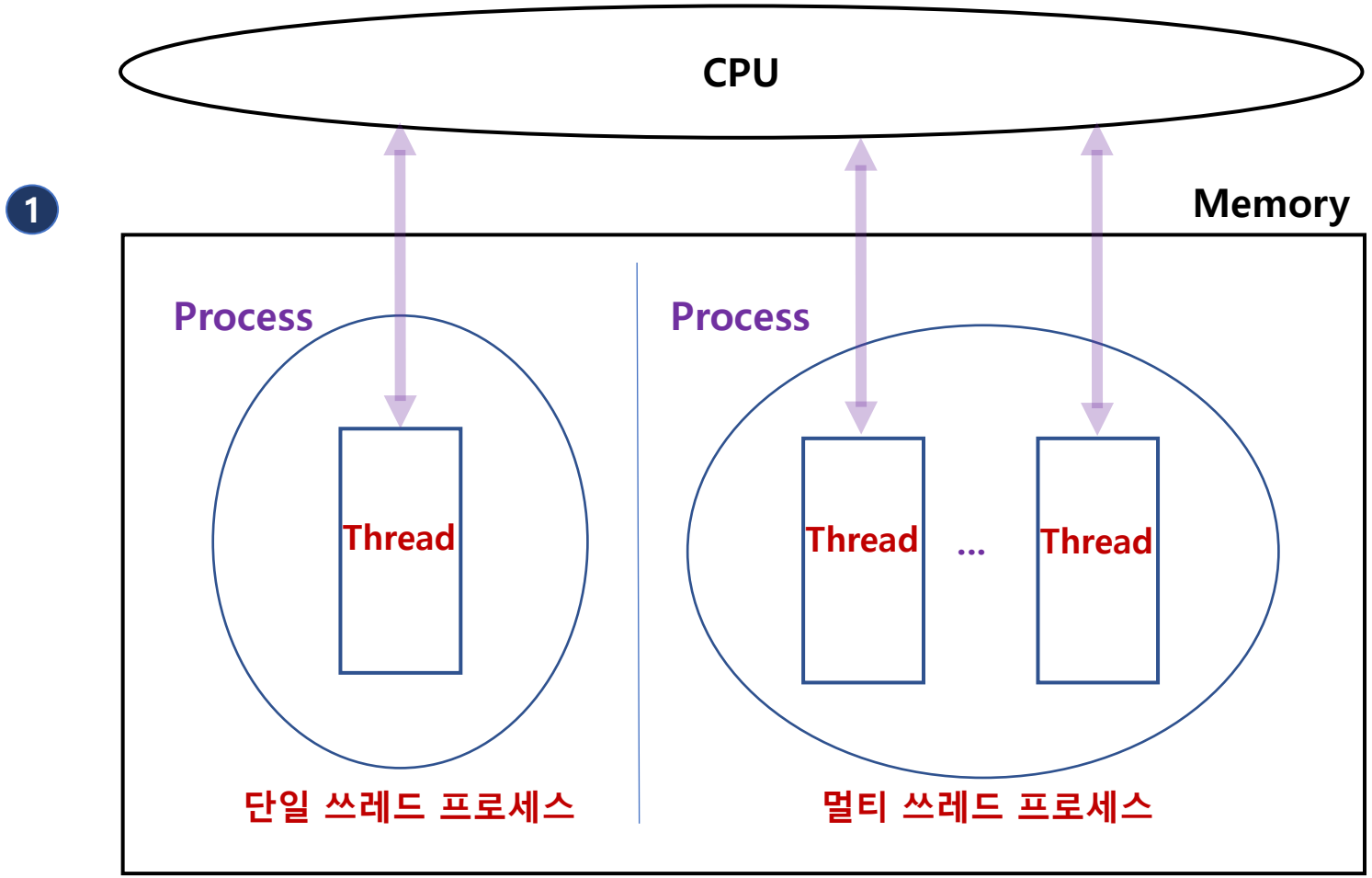
CPU는 Memory와만 이야기

6 **Q. Loading을 두 번 실행하면 (한글파일을 두 개 띄우면)?**

→ 멀티 프로세스(multi-process)

Program vs. Process vs. Thread의 개념

☞ Process의 구조



3 **Q. Thread가 없는 Process가 있을까?**

= CPU를 사용하지 않는 프로그램

2

CPU는 Memory와만 이야기

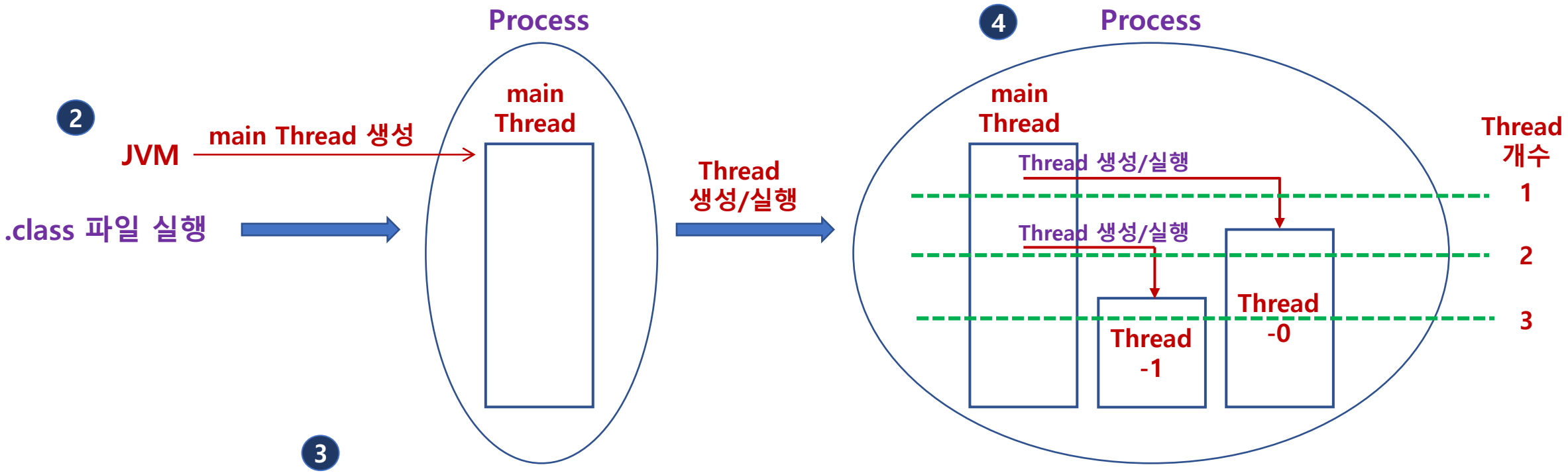
CPU는 Memory 내의 Process와만 이야기

CPU는 Memory내 Process 속에 있는 Thread와만 이야기

Thread는 CPU를 사용하는 최소단위

Program vs. Process vs. Thread의 개념

1 ☞ Java Program상의 Thread



Multi-Thread의 필요성

Multi-Thread의 필요성

☞ Multi-Thread의 필요성

1
- **비디오 프레임 번호** (1, 2, 3, 4, 5)
- **자막번호** (하나, 둘, 셋, 넷, 다섯)

원하는 결과

2
```
(비디오 프레임) 1 – (자막) 하나
(비디오 프레임) 2 – (자막) 둘
(비디오 프레임) 3 – (자막) 셋
(비디오 프레임) 4 – (자막) 넷
(비디오 프레임) 5 – (자막) 다섯
```

3
```
public static void main(String[] ar) {

  //(비디오프레임번호) 1~5 저장
  int[] intArray = new int[] {1,2,3,4,5};
  //(자막 번호) 하나~다섯 저장
  String[] strArray = new String[] {"하나","둘","셋","넷","다섯"};

  for (int i=0; i<intArray.length; i++) {//(비디오프레임번호) 1~5 출력
    System.out.println("(비디오 프레임) " + intArray[i]);
    try { Thread.sleep(200); } catch (InterruptedException e) { }
  }
4 for (int i=0; i<strArray.length; i++) {//(자막 번호) 하나~다섯 출력
    System.out.println("(자막) "+ strArray[i]);
    try { Thread.sleep(200); } catch (InterruptedException e) { }
  }

}
```

0.2초(200ms)동안 일시정지

5
```
(비디오 프레임) 1
(비디오 프레임) 2
(비디오 프레임) 3
(비디오 프레임) 4
(비디오 프레임) 5
(자막) 하나
(자막) 둘
(자막) 셋
(자막) 넷
(자막) 다섯
```

즉, 영화 끝나고 자막나오기 시작

↓

동시에 진행될 필요성

↓ **해결책**

Multi-Thread

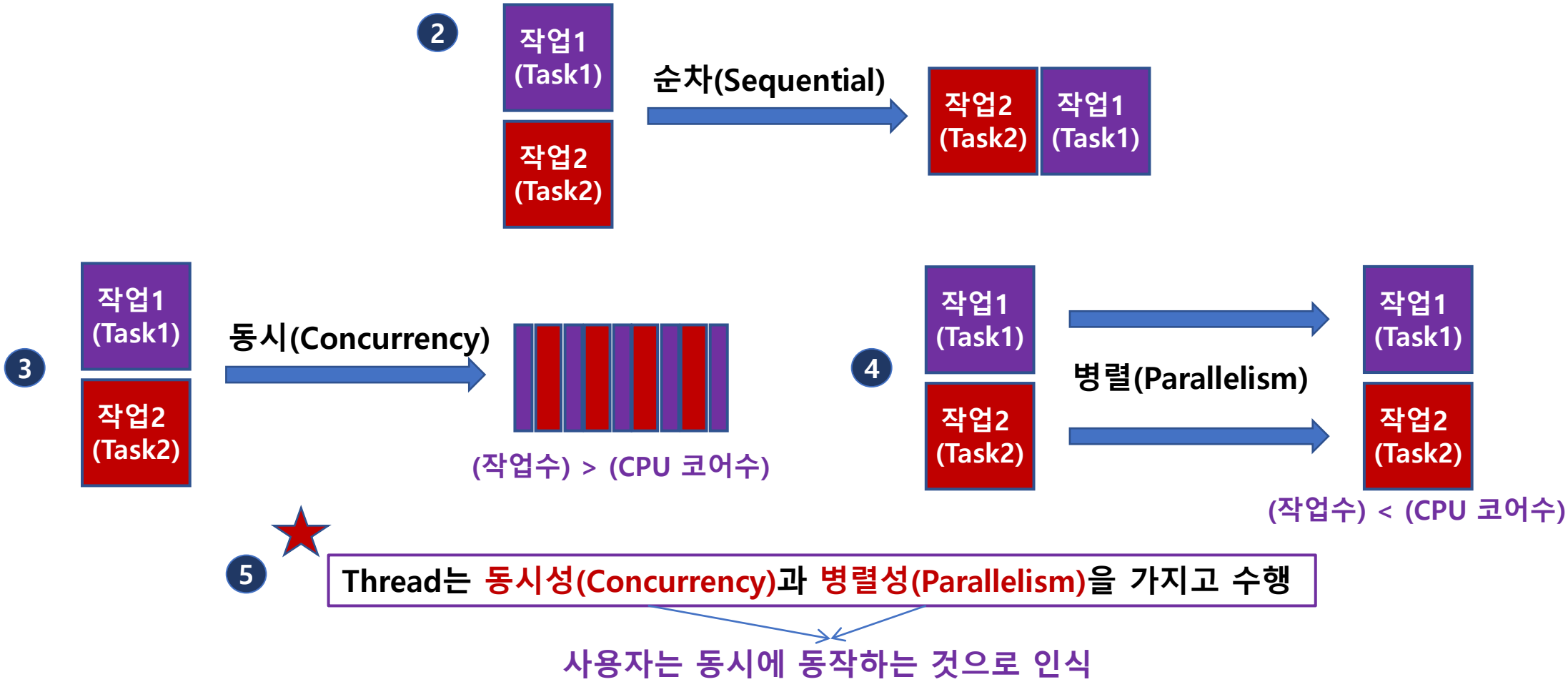
Multi-Thread의 필요성

1 ☞ Thread는 정말 동시에 수행될까?

Quiz

6

1. 작업이 3개, 코어가 4개인 경우?
2. 작업이 4개, 코어가 1개인 경우?
3. 작업이 6개, 코어가 2개인 경우?



The End

Thread의 생성 및 실행방법

Thread의 생성 및 실행방법

1 ☞ Thread **생성**방법

2

생성방법#1 - **Thread** class를 **상속**받아 run() 메서드 재정의

생성방법#2 - **Runnable** interface **구현** (추상메서드(run()) 구현)

⬇

- Thread 생성자로 **Runnable 객체 전달**

3 ☞ Thread **실행**방법

4 **실행방법** - **Thread** 객체내의 **start()** 메서드 호출

5

주의 1

재정의한 메서드는 run()이지만
Thread의 실행은 **start()**메서드 호출

주의 2

Thread 객체는 재사용 할 수 없음
(→ 하나의 객체는 한번만 start() 가능)

Thread의 생성 및 실행방법

1 ☞ Thread 생성/실행 **방법#1** - **Thread** class를 **상속**받아 run() 메서드 재정의

2 **STEP1** 클래스정의

Thread class를 **상속**받아 run() 메서드 override한 **클래스 정의 (또는 익명클래스)**

```
class MyThread extends Thread{
  @Override
  public void run() {
    //쓰레드 작업내용
  }
}
```

3 **STEP2** 객체생성

Thread **객체 생성**

```
Thread myThread = new MyThread();
또는
MyThread myThread = new MyThread();
```

4 **STEP3** Thread 실행

start() 메서드를 이용하여 Thread 실행

```
myThread.start();
```

5 **Q.** run()을 재정의했지만 start()로 Thread를 실행하는 이유?

start() = **새로운 쓰레드 생성/추가를 위한 모든 준비** + **새로운 쓰레드위에 run() 실행**

6 **run() 호출시 단일쓰레드로 동작**

run()을 호출해도 오류는 없지만 **단일 쓰레드** 내에서 동작

☞ Thread 생성/실행 방법#1 - Thread class를 상속받아 run() 메서드 재정의

1 **방법#1. CASE1 (M1C1)** **Thread(main, SMIFileThread)**

```
class SMIFileThread extends Thread {
  @Override
  public void run() {

    // (자막 번호) 하나~다섯 저장 출력
    String[] strArray = new String[]{"하나","둘","셋","넷","다섯"};
    try {Thread.sleep(10);} catch (InterruptedException e1) {}
                          단위: ms
    for (int i = 0; i < strArray.length; i++) {
      System.out.println("(자막) " + strArray[i]);
      try {Thread.sleep(200);} catch (InterruptedException e) {}
    }
  }
}
```

3

4 자막이 약간 늦게 나오게 하기 위해서 추가

```
public static void main(String[] args) {

  //SMIFileThread 생성 및 실행
  Thread smiFileThread = new SMIFileThread();
  smiFileThread.start();

  //(비디오프레임번호) 1~5 저장 + 출력
  int[] intArray = new int[] {1,2,3,4,5};

  for (int i=0; i<intArray.length; i++) {
    System.out.print("(비디오 프레임) " + intArray[i]+"-");
      try { Thread.sleep(200); } catch (InterruptedException e) { }
  }
}
```

2

5

```
(비디오 프레임) 1-(자막) 하나
(비디오 프레임) 2-(자막) 둘
(비디오 프레임) 3-(자막) 셋
(비디오 프레임) 4-(자막) 넷
(비디오 프레임) 5-(자막) 다섯
```

Thread의 생성 및 실행방법

1 **방법#1. CASE2 (M1C2)** **Thread(main, SMIFileThread, VideoFileThread)**

```
class SMIFileThread extends Thread {
  @Override
  public void run() {

    // (자막 번호) 하나~다섯 저장 출력
    String[] strArray = new String[]{"하나","둘","셋","넷","다섯"};
    try {Thread.sleep(10);} catch (InterruptedException e1) {}

    for (int i = 0; i < strArray.length; i++) {
      System.out.println("(자막) " + strArray[i]);
      try {Thread.sleep(200);} catch (InterruptedException e) {}
    }
  }
}
```

2

```
class VideoFileThread extends Thread {
  @Override
  public void run() {

    //(비디오프레임번호) 1~5 저장 + 출력
    int[] intArray = new int[] {1,2,3,4,5};

    for (int i=0; i<intArray.length; i++) {
      System.out.print("(비디오 프레임) " + intArray[i]+"-");
      try { Thread.sleep(200); } catch (InterruptedException e) { }
    }
  }
}
```

3

```
public static void main(String[] ar) {

  //Thread 생성
  Thread videoFileThread =
                                  new VideoFileThread();
  Thread smiFileThread = new SMIFileThread();

  //Thread 실행
  videoFileThread.start();
  smiFileThread.start();
}
```

4

```
(비디오 프레임) 1-(자막) 하나
(비디오 프레임) 2-(자막) 둘
(비디오 프레임) 3-(자막) 셋
(비디오 프레임) 4-(자막) 넷
(비디오 프레임) 5-(자막) 다섯
```

Thread의 생성 및 실행방법

1 ☞ Thread 생성/실행 **방법#2** - **Runnable** interface **구현** → Thread 생성자로 **Runnable 객체 전달**

2 **STEP1** **클래스정의**

Runnable interface를 **구현**한 **클래스 정의**
(추상메서드 **run()**구현)

```
class MyRunnable implements Runnable{
  @Override
  public void run() {
    //쓰레드 작업내용
  }
}
```

3 **STEP2** **객체생성**

A. Runnable 객체 생성
B. Thread 객체 생성(생성자에 Runnable 객체 전달)

```
Runnable r = new MyRunnable();
          또는
MyRunnable r = new MyRunnable();

Thread myThread = new Thread(r);
```

4 **STEP3** **Thread 실행**

start() 메서드를 이용하여 Thread 실행

```
myThread.start();
```

5 **Q.** Runnable 객체에서 바로 start() 하지 않는 이유?

→ Runnable은 **함수적 인터페이스**로 내부에 **start() 메서드가 없어** start()를 위해서 Thread 객체가 필요함

Thread의 생성 및 실행방법

1 **방법#2. CASE1 (M2C1)** **Thread(main, SMIFileRunnable)**

```
class SMIFileRunnable implements Runnable {
  @Override
  public void run() {

    // (자막 번호) 하나~다섯 저장 출력
    String[] strArray = new String[]{"하나","둘","셋","넷","다섯"};
    try {Thread.sleep(10);} catch (InterruptedException e1) {}

    for (int i = 0; i < strArray.length; i++) {
      System.out.println("(자막) " + strArray[i]);
      try {Thread.sleep(200);} catch (InterruptedException e) {}
    }
  }
}
```

2

```
public static void main(String[] args) {

    //SMIFileRunnable 생성 및 실행
    Runnable smiFileRunnable = new SMIFileRunnable();
    //smiFileRunnable.start(); //오류
    Thread thread = new Thread(smiFileRunnable);
    thread.start();

    //(비디오프레임번호) 1~5 저장 + 출력
    int[] intArray = new int[] {1,2,3,4,5};
    for (int i=0; i<intArray.length; i++) {
    System.out.print("(비디오 프레임) " + intArray[i]+"-");
      try { Thread.sleep(200); } catch (InterruptedException e) { }
    }
}
```

3

```
(비디오 프레임) 1-(자막) 하나
(비디오 프레임) 2-(자막) 둘
(비디오 프레임) 3-(자막) 셋
(비디오 프레임) 4-(자막) 넷
(비디오 프레임) 5-(자막) 다섯
```

Thread의 생성 및 실행방법

1 **방법#2. CASE2 (M2C2)** **Thread(main, SMIFileRunnable, VideoFileRunnable)**

```
class SMIFileRunnable implements Runnable {
  @Override
  public void run() {

    // (자막 번호) 하나~다섯 저장 출력
    String[] strArray = new String[]{"하나","둘","셋","넷","다섯"};
    try {Thread.sleep(10);} catch (InterruptedException e1) {}

    for (int i = 0; i < strArray.length; i++) {
      System.out.println("(자막) " + strArray[i]);
      try {Thread.sleep(200);} catch (InterruptedException e) {}
    }
  }
}
```

2

```
class VideoFileRunnable implements Runnable {
  @Override
  public void run() {

     //(비디오프레임번호) 1~5 저장 + 출력
    int[] intArray = new int[] {1,2,3,4,5};

    for (int i=0; i<intArray.length; i++) {
      System.out.print("(비디오 프레임) " + intArray[i]+"-");
      try { Thread.sleep(200); } catch (InterruptedException e) { }
    }
  }
}
```

3

```
public static void main(String[] ar) {

  //Runnable 생성
  Runnable smiFileRunnable =
                              new SMIFileRunnable();
  Runnable videoFileRunnable =
                            new VideoFileRunnable();

  //Thread 생성
  Thread myThread1 =
                         new Thread(smiFileRunnable);
  Thread myThread2 =
                       new Thread(videoFileRunnable);

  //Thread 실행
  myThread1.start();
  myThread2.start();

}
```

4

```
(비디오 프레임) 1-(자막) 하나
(비디오 프레임) 2-(자막) 둘
(비디오 프레임) 3-(자막) 셋
(비디오 프레임) 4-(자막) 넷
(비디오 프레임) 5-(자막) 다섯
```

Thread의 생성 및 실행방법

1 **방법#2. CASE3 (M2C3)** **Thread(main, SMIFileRunnable, VideoFileRunnable) / 익명이너클래스**

```
public static void main(String[] args) {

  // Thread 생성 1

  Thread myThread1 = new Thread(new Runnable() {

    @Override
    public void run() {

      // (자막 번호) 하나~다섯 저장 출력
      String[] strArray =
                    new String[] {"하나","둘","셋","넷","다섯" };
      try {Thread.sleep(10);}
      catch (InterruptedException e1) {}

      for (int i = 0; i < strArray.length; i++) {
        System.out.println("(자막) " + strArray[i]);
        try {Thread.sleep(200);}
        catch (InterruptedException e) {}
      }
    }
  });
```

2

```
  // Thread 생성 2

  Thread myThread2 = new Thread(new Runnable() {
    @Override
    public void run() {

      // (비디오프레임번호) 1~5 저장 + 출력
      int[] intArray = new int[] { 1, 2, 3, 4, 5 };

      for (int i = 0; i < intArray.length; i++) {
        System.out.print("(비디오 프레임) "+intArray[i]+"-");
        try {Thread.sleep(200);}
        catch (InterruptedException e) {}
      }
    }
  });

  // Thread 실행
  myThread1.start();
  myThread2.start();

}
```

3

```
(비디오 프레임) 1-(자막) 하나
(비디오 프레임) 2-(자막) 둘
(비디오 프레임) 3-(자막) 셋
(비디오 프레임) 4-(자막) 넷
(비디오 프레임) 5-(자막) 다섯
```

The End

Thread의 속성
(Thread 객체 가져오기/이름/
Thread 개수/우선순위/데몬설정)

Thread의 속성 (**Thread 객체 가져오기/이름/Thread 개수**/우선순위/데몬설정)

☞ 현재 Thread 객체 참조 (Thread 클래스의 **정적**메서드)

1

```
static Thread Thread.currentThread()
```

→ **Thread 참조 객체가 없는 경우 사용**

☞ 실행중인 Thread 개수 가져오기 (Thread 클래스의 **정적**메서드)

2

```
static int Thread.activeCount()
```

→ **같은 ThreadGroup내의 active thread 수**

생성된 쓰레드는 자신을 생성한 쓰레드와 같은 그룹에 속함

☞ Thread 이름 설정 및 가져오기 (Thread 클래스의 **인스턴스**메서드)

```
String setName(String name)
```

→ **Thread의 이름 설정하기**

```
String getName()
```

→ **Thread의 이름 가져오기**

3

쓰레드의 이름을 지정하지 않는 경우 **thread-0, thread-1, … thread-N**과 같이 번호를 하나씩 증가시키면서 이름 자동 부여

Thread의 속성 (**Thread 객체 가져오기/이름/Thread 개수**/우선순위/데몬설정)

☞ Thread 객체 가져오기/이름/Thread 개수

1

```
public static void main(String[] ar) {

  //#1. 객체 가져오기 + 쓰레드 수
  Thread curThread = Thread.currentThread();
  System.out.println(
          "현재 쓰레드 이름:"+curThread.getName());
  System.out.println(
                "쓰레드 수="+Thread.activeCount());

  //#2. 쓰레드 이름 자동 설정
  for (int i = 0; i < 3; i++) {
    Thread thread = new Thread();
    System.out.println(thread.getName());
    thread.start();
  }

  // #3. 쓰레드 이름 직접 설정
  for (int i = 0; i < 3; i++) {
    Thread thread = new Thread();
    thread.setName(i+"번째 쓰레드");
    System.out.println(thread.getName());
    thread.start();
  }
```

2

```
  //#4. 쓰레드 이름 자동 설정
  for (int i = 0; i < 3; i++) {
    Thread thread = new Thread();
    System.out.println(thread.getName());
    thread.start();
  }

  //#5. 쓰레드 수
  System.out.println("쓰레드 수="+Thread.activeCount());
}
```

```
현재 쓰레드 이름:main
쓰레드 수=1
Thread-0
Thread-1
Thread-2
0번째 쓰레드
1번째 쓰레드
2번째 쓰레드
Thread-6
Thread-7
Thread-8
쓰레드 수=4
```

Thread의 속성 (Thread 객체 가져오기/이름/Thread 개수/**우선순위**/데몬설정)

☞ Thread 객체 우선순위 가져오기 (Thread 클래스의 **인스턴스**메서드)

1

```
int getPriority ()
```

→ **Thread의 우선순위(Priority) 가져오기**

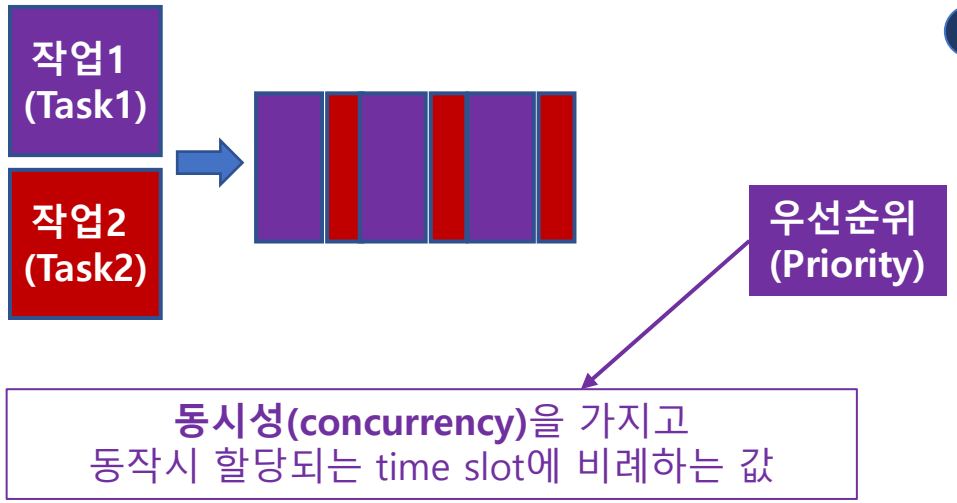
☞ Thread 객체 우선순위 정하기 (Thread 클래스의 **인스턴스**메서드)

2

```
void setPriority (int priority)
```

→ **Thread의 우선순위(Priority) 설정하기**

4



3

5

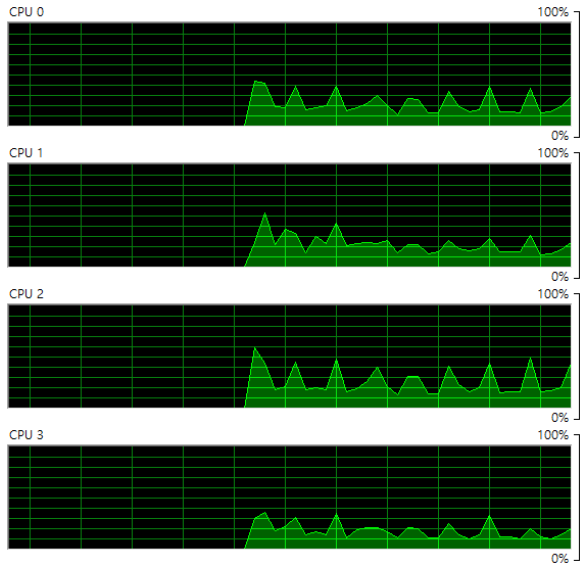
참고 현재 시스템의 CPU코어수 가져오기 — `Runtime.getRuntime().availableProcessors()`

Thread의 속성 (Thread 객체 가져오기/이름/Thread 개수/**우선순위**/데몬설정)

1

```
class MyThread extends Thread{
  @Override
  public void run() {
    //#약간의 시간 지연
    for(long i=0; i<100000000; i++) {}
    System.out.println(
        getName() + " 우선순위: " + getPriority());
  }
}
```

3




```
코어수 : 4
Thread-2 우선순위:5
Thread-0 우선순위:5
Thread-1 우선순위:5
9 번째 쓰레드 우선순위:10
3 번째 쓰레드 우선순위:5
0 번째 쓰레드 우선순위:5
1 번째 쓰레드 우선순위:5
2 번째 쓰레드 우선순위:5
5 번째 쓰레드 우선순위:5
8 번째 쓰레드 우선순위:5
6 번째 쓰레드 우선순위:5
7 번째 쓰레드 우선순위:5
4 번째 쓰레드 우선순위:5
```

```
public static void main(String[] args) {
  //#참고. CPU Core 수 가져오기
  System.out.print("코어수 : ");
  System.out.println(
       Runtime.getRuntime().availableProcessors());

  //#1. 디폴트(default) 우선순위
  for (int i = 0; i < 3; i++) {
    Thread thread = new MyThread();
    thread.start();
  }
  try { Thread.sleep(1000); }
  catch (InterruptedException e) {}

  //#2. 우선순위 지정
  for (int i = 0; i < 10; i++) {
    Thread thread = new MyThread();
    thread.setName(i + " 번째 쓰레드");
    if (i == 9)
      thread.setPriority(Thread.MAX_PRIORITY);
      thread.start();
  }
}
```

2

4

5

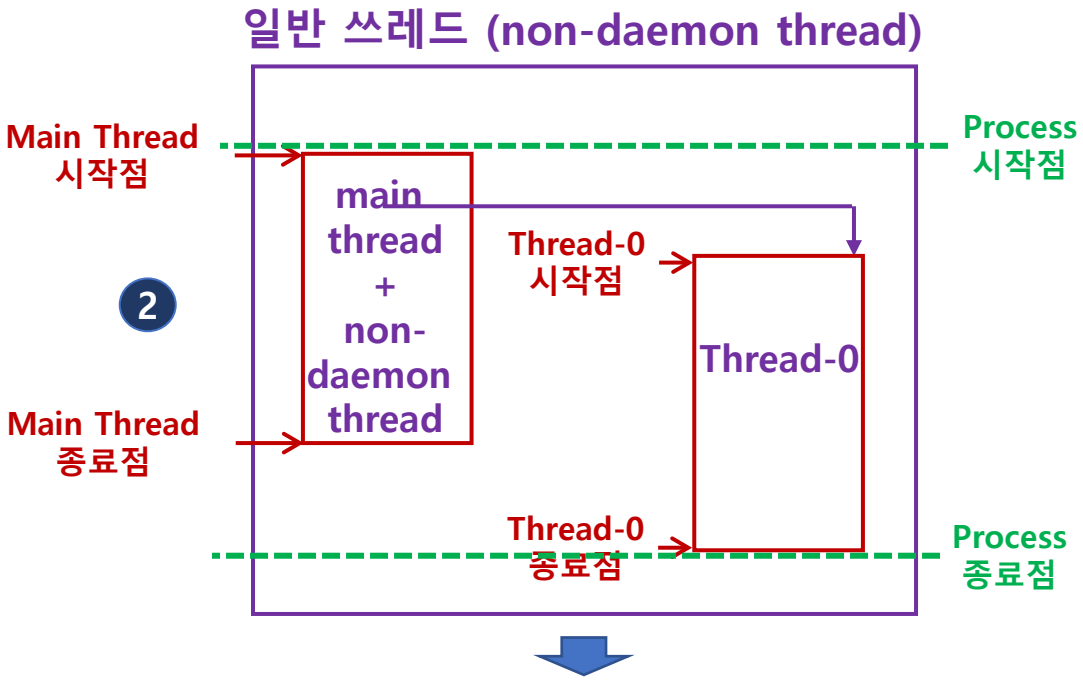
Thread의 속성 (Thread 객체 가져오기/이름/Thread 개수/우선순위/**데몬설정**)

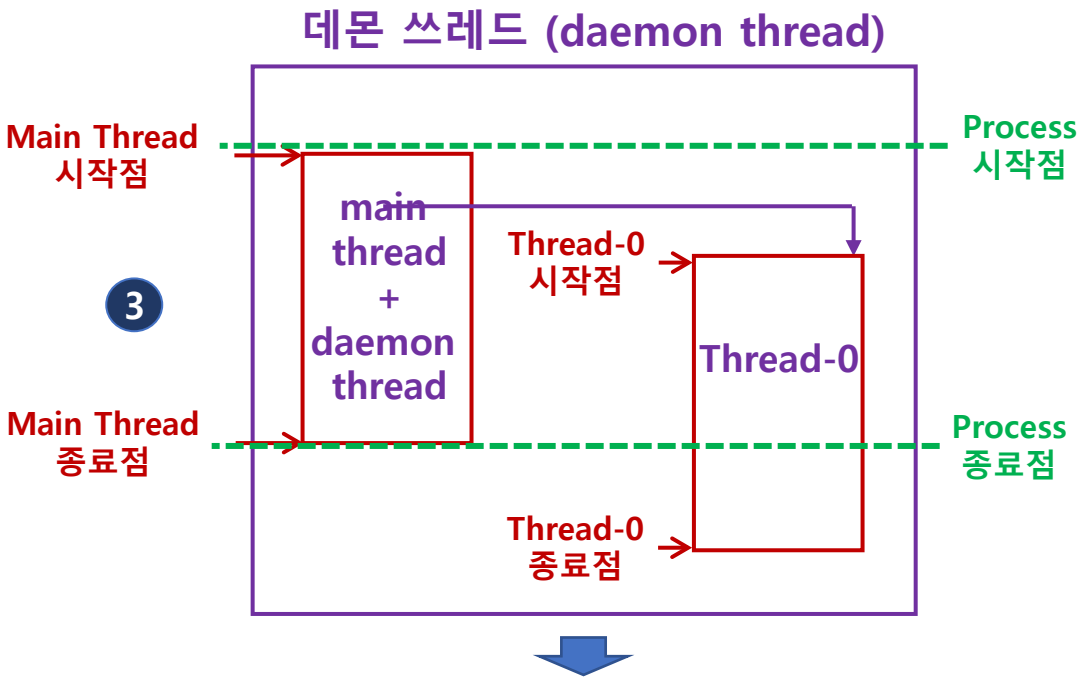
☞ Thread 데몬 설정 (Thread 클래스의 **인스턴스**메서드) → **주의**. setDeamon()은 반드시 start() 전에 호출

1

```
void setDeamon(boolean on)
```

→ **on이 true인 경우 Deamon Thread**
Default 값은 false : 일반 Thread





일반쓰레드는 다른쓰레드 종료여부와 상관없이 자신의 쓰레드가 종료되어야 프로세스 종료

데몬쓰레드는 일반쓰레드(사용자쓰레드)가 **모두 종료**되면 작업이 완료되지 않았어도 함께 종료

Thread의 속성 (Thread 객체 가져오기/이름/Thread 개수/우선순위/**데몬설정**)

```
class MyThread extends Thread {
  @Override
  public void run() {
    System.out.println(getName() + ":" + (isDaemon() ? "데몬쓰레드" : "일반쓰레드"));

    for (int i = 0; i < 6; i++) {
      System.out.println(getName() + ":" + i + "초");
      try {Thread.sleep(1000);}
      catch (InterruptedException e) {}
    }

  }
}
```

1

```
thread1:일반쓰레드
thread1:0초
thread1:1초
thread1:2초
main thread 종료
thread1:3초
thread1:4초
thread1:5초
```

```
public static void main(String[] args) {

  //#1. 일반쓰레드
  Thread thread1 = new MyThread();
  thread1.setDaemon(false);
  thread1.setName("thread1");
  thread1.start();

  ////#2. 데몬쓰레드
  //Thread thread2 = new MyThread();
  //thread2.setDaemon(true);
  //thread2.setName("thread2");
  //thread2.start();

  //#3. 3초후 MainThread 종료
  try { Thread.sleep(3000); }
  catch (InterruptedException e) {}
  System.out.println("main thread 종료");
}
```

2

Thread의 속성 (Thread 객체 가져오기/이름/Thread 개수/우선순위/**데몬설정**)

```
class MyThread extends Thread {
  @Override
  public void run() {
    System.out.println(getName() + ":" + (isDaemon() ? "데몬쓰레드" : "일반쓰레드"));

    for (int i = 0; i < 6; i++) {
      System.out.println(getName() + ":" + i + "초");
      try {Thread.sleep(1000);}
      catch (InterruptedException e) {}
    }

  }
}
```

1

```
thread2:데몬쓰레드
thread2:0초
thread2:1초
thread2:2초
main thread 종료
```

```
public static void main(String[] args) {

  ////#1. 일반쓰레드
  //Thread thread1 = new MyThread();
  //thread1.setDaemon(false);
  //thread1.setName("thread1");
  //thread1.start();

  //#2. 데몬쓰레드
  Thread thread2 = new MyThread();
  thread2.setDaemon(true);
  thread2.setName("thread2");
  thread2.start();

  //#3. 3초후 MainThread 종료
  try { Thread.sleep(3000); }
  catch (InterruptedException e) {}
  System.out.println("main thread 종료");
}
```

2

Thread의 속성 (Thread 객체 가져오기/이름/Thread 개수/우선순위/**데몬설정**)

```
class MyThread extends Thread {
  @Override
  public void run() {
    System.out.println(getName() + ":" + (isDaemon() ? "데몬쓰레드" : "일반쓰레드"));

    for (int i = 0; i < 6; i++) {
      System.out.println(getName() + ":" + i + "초");
      try {Thread.sleep(1000);}
      catch (InterruptedException e) {}
    }

  }
}
```

1

```
public static void main(String[] args) {

  //#1. 일반쓰레드
  Thread thread1 = new MyThread();
  thread1.setDaemon(false);
  thread1.setName("thread1");
  thread1.start();

  //#2. 데몬쓰레드
  Thread thread2 = new MyThread();
  thread2.setDaemon(true);
  thread2.setName("thread2");
  thread2.start();

  //#3. 3초후 MainThread 종료
  try { Thread.sleep(3000); }
  catch (InterruptedException e) {}
  System.out.println("main thread 종료");
}
```

2

```
thread1:일반쓰레드
thread2:데몬쓰레드
thread1:0초
thread2:0초
thread1:1초
thread2:1초
thread2:2초
thread1:2초
main thread 종료
thread1:3초
thread2:3초
thread1:4초
thread2:4초
thread2:5초
thread1:5초
```

3

데몬 쓰레드는 main쓰레드를 포함해서 모든 일반쓰레드가 종료해야 함께 종료됨

The End

Thread 동기화

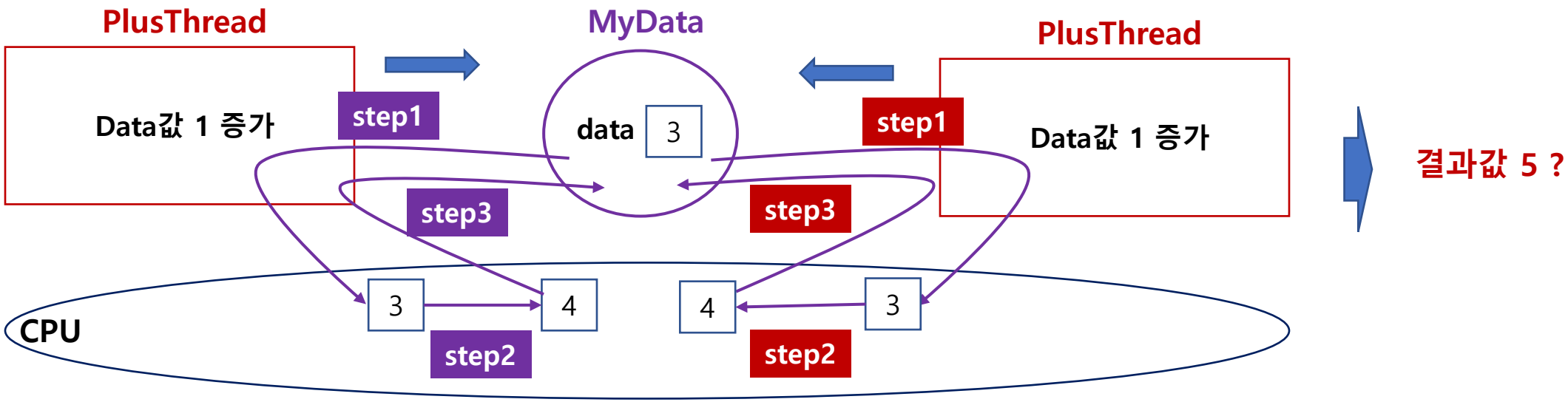
Thread 동기화

☞ **동기화(synchronized)**의 의미

1
- 하나의 작업이 완전히 **완료된 후** 다른 작업 수행
- cf. **비동기식**: 하나의 작업 명령 이후(완료와 상관없이) 바로 다른 작업 명령을 수행

☞ **동기화(synchronized)**의 필요성 - 두 개의 Thread가 하나의 객체를 공유하는 경우

2



3 **step3** 보다 **step1** 이 먼저 발생하면?

결과값 4

Thread 동기화 ☞ 동기화(synchronized)의 필요성 - 두 개의 Thread가 하나의 객체를 공유하는 경우

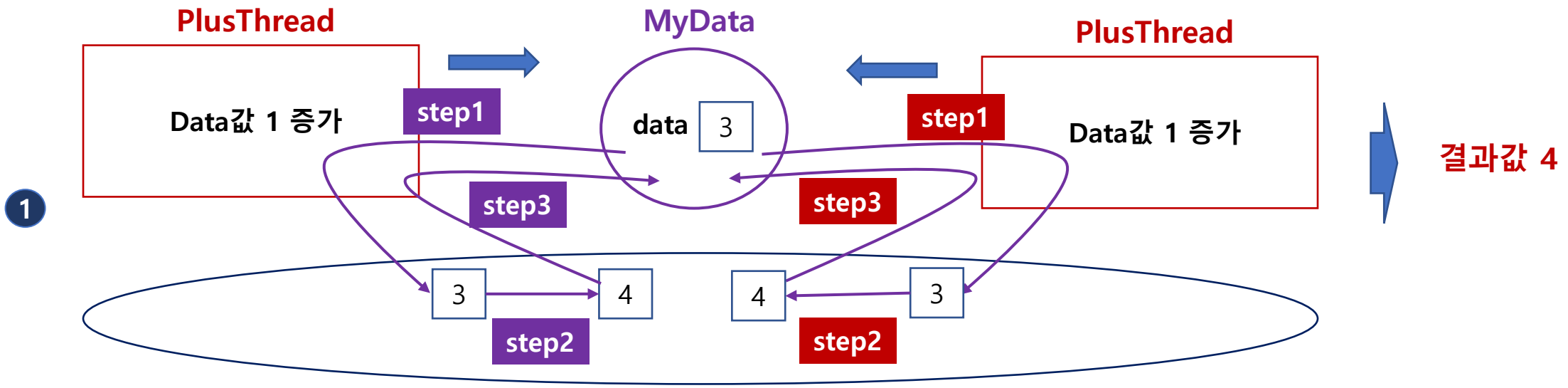
1
```
class MyData {
  int data = 3;

  public void plusData() {
    //데이터를 바로 가져와 2초 뒤 결과값 저장
    int mydata = data;
    try { Thread.sleep(2000);
    } catch (InterruptedException e) {}
    data=mydata+1;
  }

}
```

2
```
class PlusThread extends Thread{
  MyData myData;
  public PlusThread(MyData myData) {
    this.myData = myData;
  }
  @Override
  public void run() {
    myData.plusData();
    System.out.println(
              getName()+"실행결과: "+myData.data);
  }
}
```

3
```
public static void main(String[] args) {

  //#공유객체
  MyData myData = new MyData();

  //#plusThread1
  Thread plusThread1 = new PlusThread(myData);
  plusThread1.setName("plusThread1");
  plusThread1.start();

  try { Thread.sleep(1000); }
  catch (InterruptedException e) {}

  //#plusThread2
  Thread plusThread2 = new PlusThread(myData);
  plusThread2.setName("plusThread2");
  plusThread2.start();

}
```

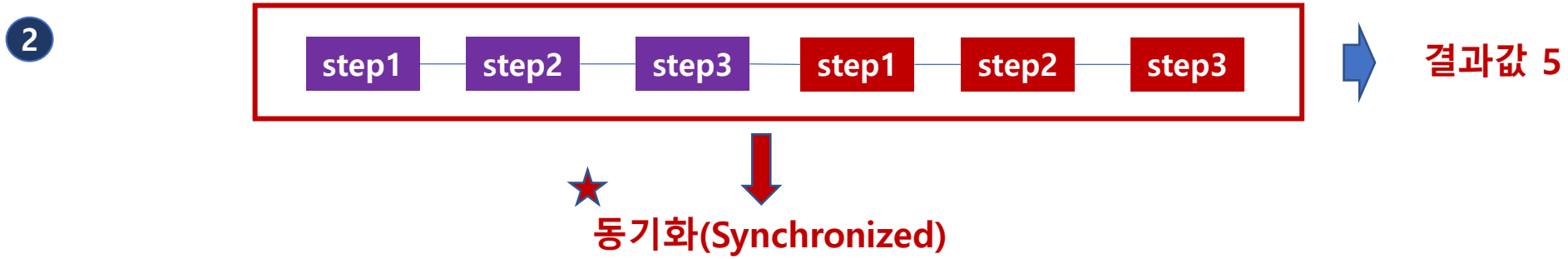
```
plusThread1 실행결과: 4
plusThread2 실행결과: 4
```

Thread 동기화



☞ 문제점의 **해결책**

- 공유객체를 한번에 하나의 Thread만 사용할 수 있도록 함 (하나의 Thread가 사용 중일때는 객체를 **lock**)
- 즉, 다음 순서로 진행



Thread 동기화

☞ 동기화(synchronized) 방법

1

방법#1 - 메서드 동기화 (synchronized method) ⟵ 동시에 두 개의 Thread가 동기화 **메서드** 사용불가

방법#2 - 블록 동기화 (synchronized block) ⟵ 동시에 두 개의 Thread가 동기화 **블록** 사용불가

☞ **방법 #1**. 메서드 동기화

- 메서드 동기화 문법

2

```
접근지정자 synchronized 리턴타입 메서드명(입력매개변수){

  //동기화가 필요한 코드

}
```

3

```
class MyData {
  int data = 3;

  public synchronized void plusData() {
    //데이터를 바로 가져와 2초 뒤 결과값 저장
    int mydata = data;
    try { Thread.sleep(2000);
    } catch (InterruptedException e) {}
    data=mydata+1;
  }

}
```

Thread 동기화

방법#1 - 메서드 동기화 (synchronized method)

동시에 두 개의 Thread가 동기화 **메서드** 사용불가

```
public static void main(String[] args) {

  //#공유객체
  MyData myData = new MyData();

  //#plusThread1
  Thread plusThread1 = new PlusThread(myData);
  plusThread1.setName("plusThread1");
  plusThread1.start();

  try { Thread.sleep(1000); }
  catch (InterruptedException e) {}

  //#plusThread2
  Thread plusThread2 = new PlusThread(myData);
  plusThread2.setName("plusThread2");
  plusThread2.start();

}
```

3

```
plusThread1 실행결과: 4
plusThread2 실행결과: 5
```

```
class MyData {
  int data = 3;

  public synchronized void plusData() {
    //데이터를 바로 가져와 2초 뒤 결과값 저장
    int mydata = data;
    try { Thread.sleep(2000);
    } catch (InterruptedException e) {}
    data=mydata+1;
  }

}
```

1

```
class PlusThread extends Thread{
  MyData myData;
  public PlusThread(MyData myData) {
    this.myData = myData;
  }
  @Override
  public void run() {
    myData.plusData();
    System.out.println(
                getName()+"실행결과: "+myData.data);
  }
}
```

2

Thread 동기화

☞ 동기화(synchronized) 방법

방법#1 - 메서드 동기화 (synchronized method) ← 동시에 두 개의 Thread가 동기화 **메서드** 사용불가

방법#2 - 블록 동기화 (synchronized block) ← 동시에 두 개의 Thread가 동기화 **블록** 사용불가

1 ☞ **방법 #2**. 블록 동기화

- **블록** 동기화 문법

2

```
synchronized (임의의 객체) {

  //동기화가 필요한 코드

}
```

Key를 가진 객체 (**모든 객체는 저마다의 Key 하나를 가지고 있음**)
일반적으로 클래스 내부에서 바로 사용할 수 있는 객체인 **this**를 사용

3

```
class MyData {
  int data = 3;

  public void plusData() {
    synchronized(this) {
      //데이터를 바로 가져와 2초 뒤 결과값 저장
      int mydata = data;
      try { Thread.sleep(2000);
      } catch (InterruptedException e) {}
      data=mydata+1;
    }
  }
}
```

Thread 동기화

방법#2 - 블록 동기화 (synchronized block)

1

동시에 두 개의 Thread가 동기화 **블록** 사용불가

```
public static void main(String[] args) {

  //#공유객체
  MyData myData = new MyData();

  //#plusThread1
  Thread plusThread1 = new PlusThread(myData);
  plusThread1.setName("plusThread1");
  plusThread1.start();

  try { Thread.sleep(1000); }
  catch (InterruptedException e) {}

  //#plusThread2
  Thread plusThread2 = new PlusThread(myData);
  plusThread2.setName("plusThread2");
  plusThread2.start();

}
```

4

```
plusThread1 실행결과: 4
plusThread2 실행결과: 5
```

```
class MyData {
  int data = 3;

  public void plusData() {
    synchronized(this) {
      //데이터를 바로 가져와 2초 뒤 결과값 저장
      int mydata = data;
      try { Thread.sleep(2000);
      } catch (InterruptedException e) {}
      data=mydata+1;
    }
  }
}
```

2

```
class PlusThread extends Thread{
  MyData myData;
  public PlusThread(MyData myData) {
    this.myData = myData;
  }
  @Override
  public void run() {
    myData.plusData();
    System.out.println(
                getName()+"실행결과: "+myData.data);
  }
}
```

3

Thread 동기화

☞ 동기화(synchronized)의 **원리**

1. **모든 객체는 단 하나의 열쇠를 가지고 있음**

동기화(synchnoized)를 사용하면 처음 사용하는 Thread가 Key객체의 Key를 가짐

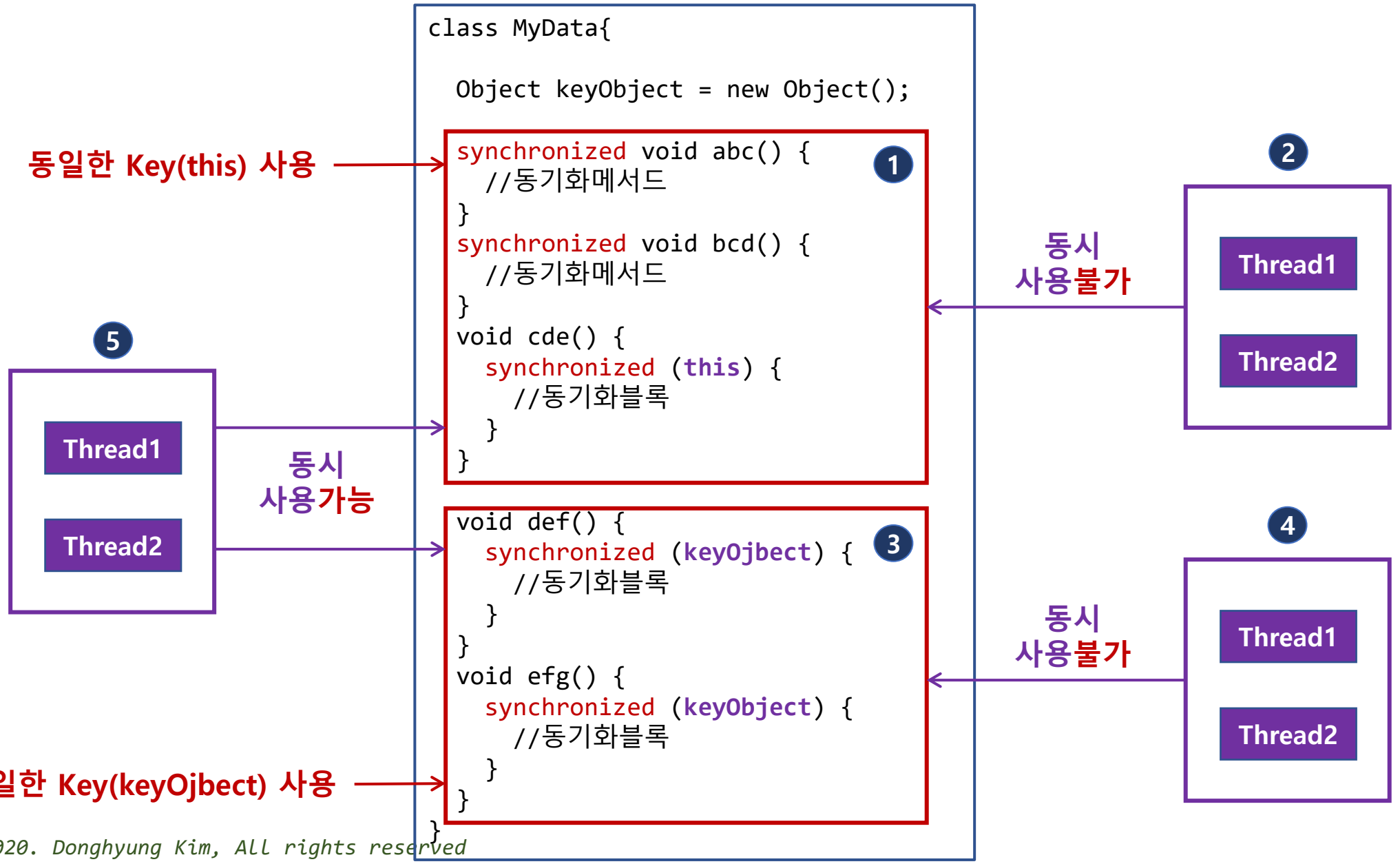
2.

동기화 메서드	동기화 블록
자기자신의 객체 (**this**)	synchronized (**key객체**){}에서 사용한 key 객체

3. **다른 Thread는 먼저 사용중인 Thread가 작업을 완료하고 Key를 반납할 때까지 대기 (Blocked)**

Thread 동기화

☞ **Key 값**에 따른 동시 사용 여부



Thread 동기화

☞ **Key 값**에 따른 동시 사용 여부

```
class MyData{
  synchronized void abc() {
    for(int i=0; i<3; i++) {
      System.out.println(i+"sec");
      try {Thread.sleep(500);}
      catch (InterruptedException e) {}
    }
  }
  synchronized void bcd() {
    for(int i=0; i<3; i++) {
      System.out.println(i+"초");
      try {Thread.sleep(500);}
      catch (InterruptedException e) {}
    }
  }
  void cde() {
    synchronized (this) {
      for(int i=0; i<3; i++) {
        System.out.println(i+"번째");
        try {Thread.sleep(500);}
        catch (InterruptedException e) {}
      }
    }
  }
}
```

1

```
public static void main(String[] ar) {
  //#공유객체
  MyData myData = new MyData();

  // #Thread 1 : abc() 실행
  new Thread() {
    public void run() {
      myData.abc();
    }
  }.start();

  // #Thread 2 : bcd() 실행
  new Thread() {
    public void run() {
      myData.bcd();
    }
  }.start();

  // #Thread 3 : cde() 실행
  new Thread() {
    public void run() {
      myData.cde();
    }
  }.start();
}
```

2

3

```
0sec
1sec
2sec
0번째
1번째
2번째
0초
1초
2초
```

Thread 동기화

```
class MyData{
  synchronized void abc() {
    for(int i=0; i<3; i++) {
      System.out.println(i+"sec");
      try {Thread.sleep(500);}
      catch (InterruptedException e) {}
    }
  }
  synchronized void bcd() {
    for(int i=0; i<3; i++) {
      System.out.println(i+"초");
      try {Thread.sleep(500);}
      catch (InterruptedException e) {}
    }
  }
  void cde() {
    synchronized (new Object()) {
      for(int i=0; i<3; i++) {
        System.out.println(i+"번째");
        try {Thread.sleep(500);}
        catch (InterruptedException e) {}
      }
    }
  }
}
```

1

```
public static void main(String[] ar) {
  //#공유객체
  MyData myData = new MyData();

  // #Thread 1 : abc() 실행
  new Thread() {
    public void run() {
      myData.abc();
    }
  }.start();

  // #Thread 2 : bcd() 실행
  new Thread() {
    public void run() {
      myData.bcd();
    }
  }.start();

  // #Thread 3 : cde() 실행
  new Thread() {
    public void run() {
      myData.cde();
    }
  }.start();
}
```

2

3

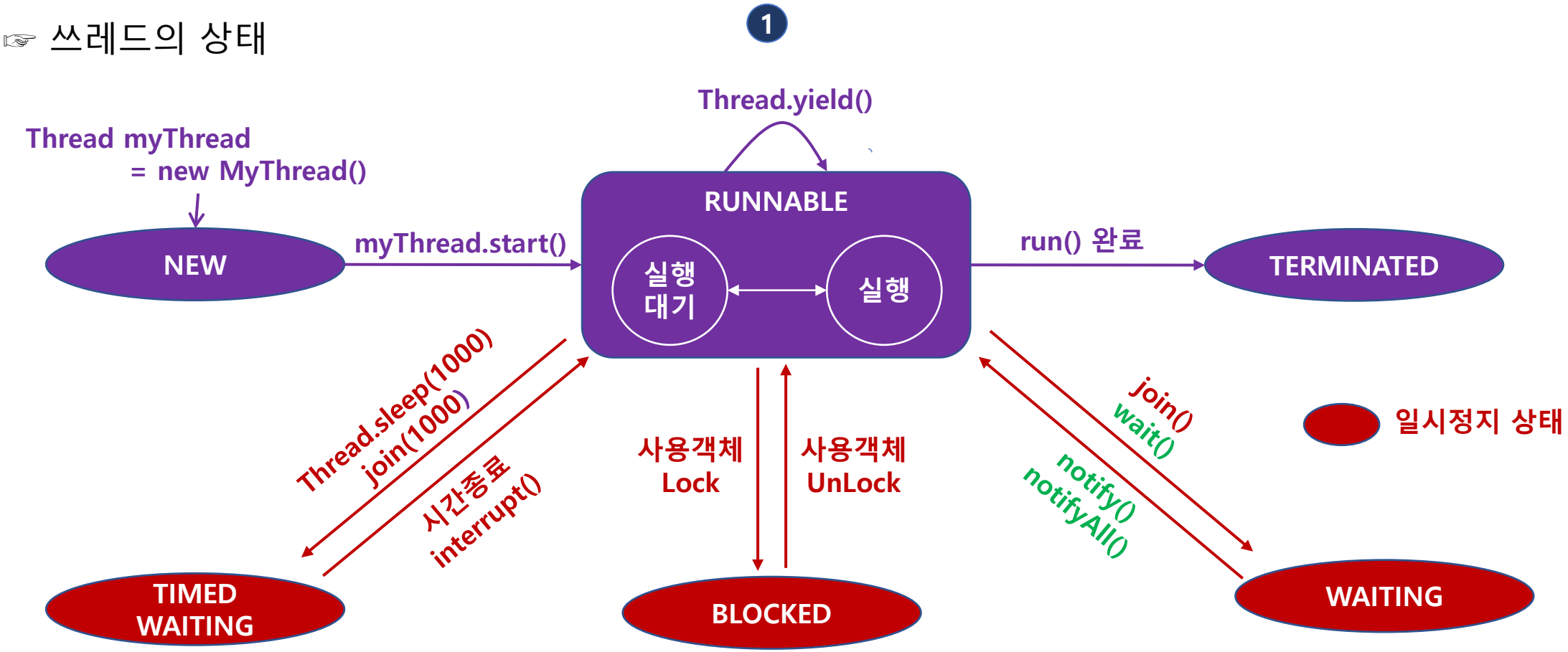
```
0sec
0번째
1sec
1번째
2sec
2번째
0초
1초
2초
```

The End

Thread 상태(State) #1

Thread 상태(State)

☞ 쓰레드의 상태



2 wait(), notify(), notifyAll()은 Object 메서드, 사용은 동기화 블록에서만 가능

Thread 상태(State)

☞ 쓰레드의 상태값 가져오기 (Thread 클래스의 **인스턴스**메서드)

1 `Thread.State getState()`

→ **Thread의 상태값을 enum 타입인 Thread.State 객체로 리턴**

사용 예시

```
State state = myThread.getState();
switch(state){
case Thread.State.NEW:
   ...
case Thread.State.RUNNABLE:
   ...
case Thread.State.TIMED_WAITING:
   ...

case Thread.State.TERMINATED:
   ...
}
```

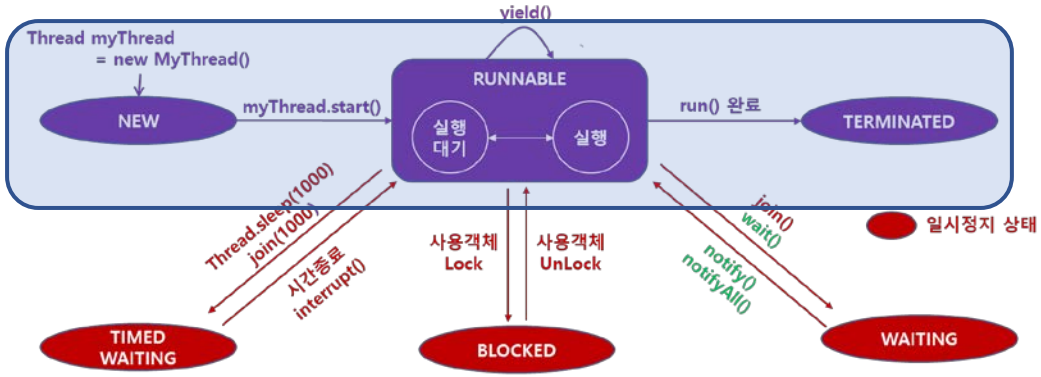
3

enum 상수

2

BLOCKED
NEW
RUNNABLE
TERMINATED
TIMED_WAITING
WAITING

Thread 상태(State)



wait(), notify(), notifyAll()은 Object 메서드, 사용은 동기화 블록에서만 가능

☞ 쓰레드의 상태 (**NEW, RUNNABLE, TERMINATE**)

1. **NEW**

new 를 통해 Thread의 객체가 생성된 상태 (start() 전)

2. **RUNNABLE**

Thread.yield() — 다른 쓰레드에게 **CPU 사용을 양보**하고 자신은 실행대기 상태로 전환

실행 대기 / 실행

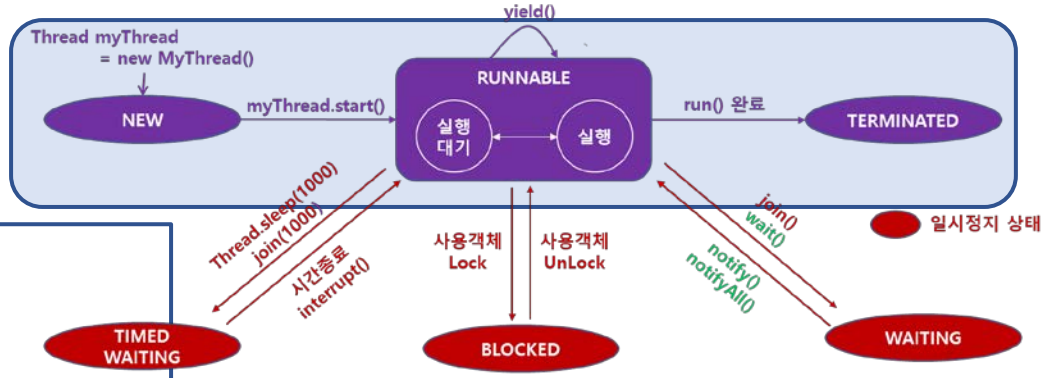
start() 이후 CPU를 사용할 수 있는 상태
다른 Thread들과 동시성(concurrency)에 따라 실행과 실행대기를 교차함

3. **TERMINATED**

run() 메서드의 작업내용이 모두 완료되어 Thread가 종료된 상태
한번 실행(start())된 Thread는 재실행은 불가하며 새롭게 객체를 생성하여야 함

Thread 상태(State)

☞ 쓰레드의 상태 (**NEW, RUNNABLE, TERMINATE**)



wait(), notify(), notifyAll()은 Object 메서드, 사용은 동기화 블록에서만 가능

```
public static void main(args) {
  //#.쓰레드 상태
  Thread.State state;

  //#1. 객체생성 (NEW)
  Thread thread = new Thread() {
    @Override
    public void run() {
      for(long i=0; i<100000000L; i++) {}
    }
  };
  state = thread.getState();
  System.out.println("thread State : " + state); //NEW

  //#2. Thread 시작 (RUNNABLE)
  thread.start();
  state = thread.getState();
  System.out.println("thread State : " + state); //RUNNABLE

  //#3. Thread 종료 (TERMINATED)
  try {thread.join();} catch (InterruptedException e) {}
  state = thread.getState();
  System.out.println("thread State : " + state); //TERMINATED
}
```

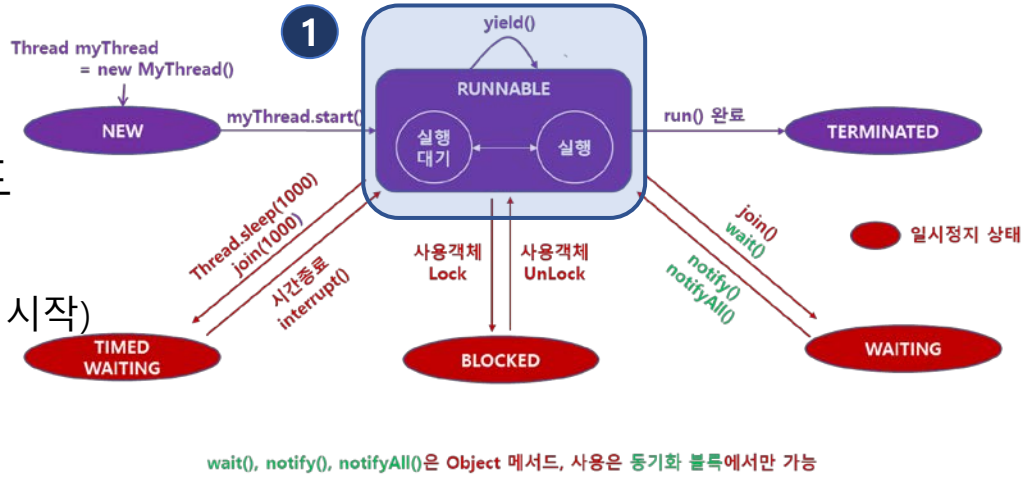
1 2 3

```
MyThread State : NEW
MyThread State : RUNNABLE
MyThread State : TERMINATED
```

Thread 상태(State)

☞ 쓰레드의 상태 (**RUNNABLE**) – Thead.yield() 정적메서드

- 다른 쓰레드에게 **CPU 사용을 양보**하고 자신은 실행대기 상태로 전환
- 다음 timeslot 위치에서는 다시 실행권한을 얻음 (yield() 위치에서 부터 시작)




```
public static void main(String[] args) {
  MyThread thread1 = new MyThread();
  thread1.setName("thread1");
  thread1.yieldFlag=false;
  thread1.setDaemon(true);
  thread1.start();

  MyThread thread2 = new MyThread();
  thread2.setName("thread2");
  thread2.yieldFlag=true;
  thread1.setDaemon(true);
  thread2.start();

  //#. 1초마다 한번씩 양보
  for(int i=0; i<6; i++) {
    try { Thread.sleep(1000); }
    catch (InterruptedException e) {}
    thread1.yieldFlag=!thread1.yieldFlag;
    thread2.yieldFlag=!thread2.yieldFlag;
  }
}
```

2

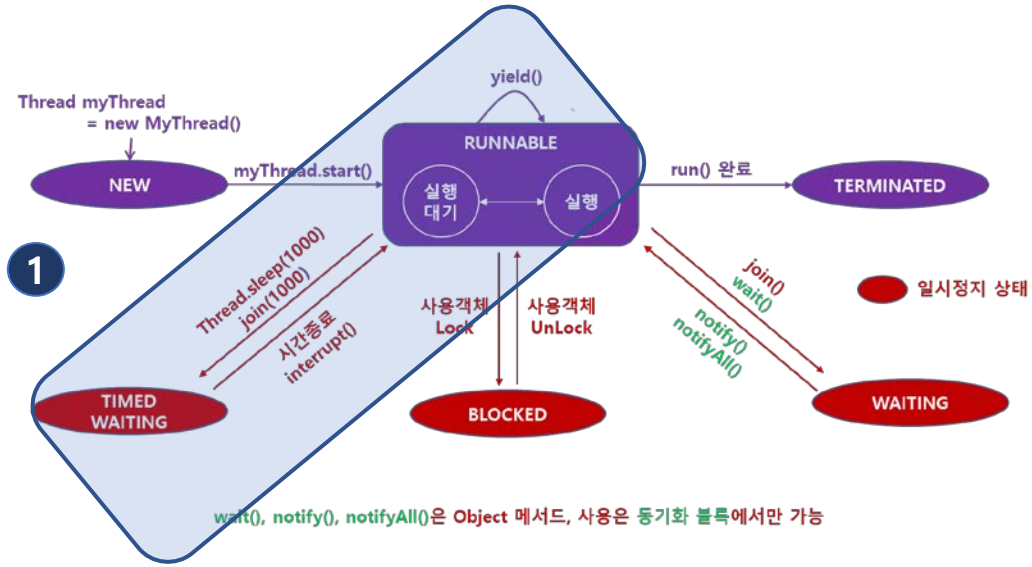
```
thread1 실행
thread1 실행
thread2 실행
thread2 실행
thread1 실행
thread1 실행
thread2 실행
thread2 실행
thread1 실행
thread1 실행
thread2 실행
thread2 실행
```

```
class MyThread extends Thread{
  boolean yieldFlag;

  @Override
  public void run() {
    while(true) {
      if(yieldFlag) {
        Thread.yield();
      } else {
        System.out.println(getName()+" 실행");
        for(long i=0; i<1000000000L; i++) {}
      }
    }
  }

}
```

3

Thread 상태(State)

☞ 쓰레드의 상태 (**TIMED_WAITING**)



1

2 sleep(시간)을 **호출한 쓰레드**가 설정한 시간동안 일시정지상태 유지

```
static void sleep(long millis)
```

RUNNABLE (실행 대기 / 실행)

Thread.sleep(1000)

join(1000)

TIMED WAITING

3

```
void join(long millis)
```

thread1.join(시간)을 호출하면 지정시간동안 thread1이 CPU를 사용하며 **호출한 쓰레드는 일시정지** 상태

일정시간 일시정지상태를 나타내며
이 상태에서는 CPU를 사용할 수 없음

정해진 시간이 종료되거나 interrupt() 발생시
다시 RUNNABLE 상태로 전환

TIMED WAITING

시간종료
interrupt()

RUNNABLE (실행 대기 / 실행)

4

```
void interrupt()
```

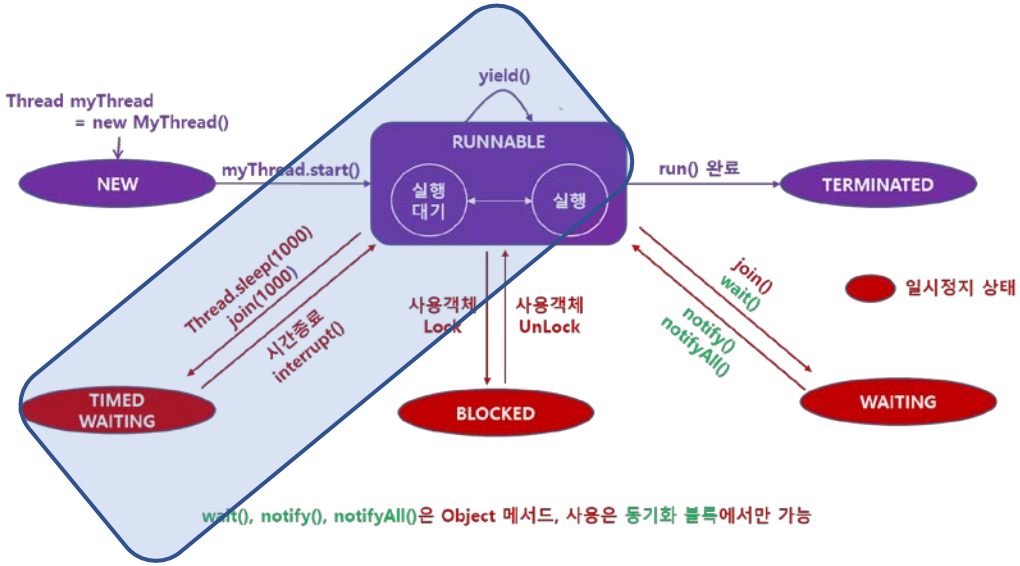
thread1.interrupt() 를 호출하면 thread1에 InterruptedException 예외를 발생시켜 일시정지 상태를 탈출함

Thread 상태(State) (TIMED_WAITING)

sleep() – interrupt()

```
class MyThread extends Thread{
  @Override
  public void run() {

    try { Thread.sleep(3000); }
    catch (InterruptedException e) {
      System.out.println(" --sleep() 중 interrupt 발생--");
      for(long i=0; i<100000000L; i++) {} //시간지연
    }
  }
}
```

1



* Interrupt()는 sleep() 또는 join() 중이어야 동작함

```
public class SleepAndInterruptExample {
  public static void main(String[] ar) throws InterruptedException {

    MyThread myThread = new MyThread();
    myThread.start();

    //#Thread.sleep()
    Thread.sleep(100); //run()이 시작되기까지 시간여유
    System.out.println("MyThread State : " + myThread.getState()); //TIMED_WAITING

    //#interrupt()
    myThread.interrupt();
    Thread.sleep(100); //예외발생까지의 시간여유
    System.out.println("MyThread State : " + myThread.getState()); //RUNNABLE
  }
}
```

2

3

```
MyThread State : TIMED_WAITING
 --sleep() 중 interrupt 발생--
MyThread State : RUNNABLE
```

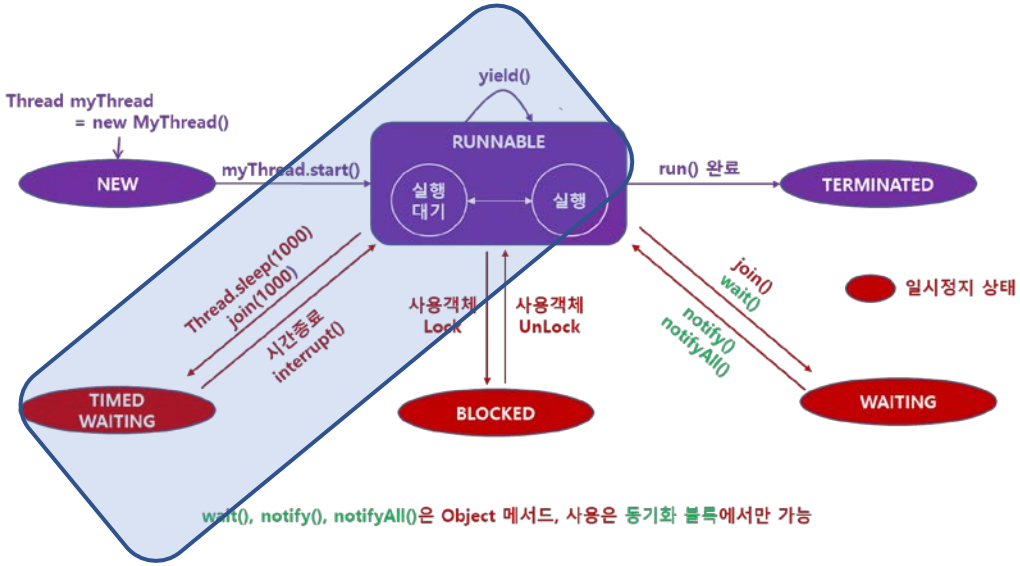
Thread 상태(State) (TIMED_WAITING)

join() – interrupt()

```
class MyThread1 extends Thread{
  @Override
  public void run() {
    for(long i=0; i<1000000000L; i++) { }
  }
}

class MyThread2 extends Thread{

  MyThread1 myThread1;

  public MyThread2(MyThread1 myThread1) {
    this.myThread1=myThread1;
  }

  @Override
  public void run() {
    try { myThread1.join(3000); }
    catch (InterruptedException e) {
      System.out.println(" --join() 중 interrupt 발생--");
    }
    for(long i=0; i<1000000000L; i++) { }
  }
}
```

1

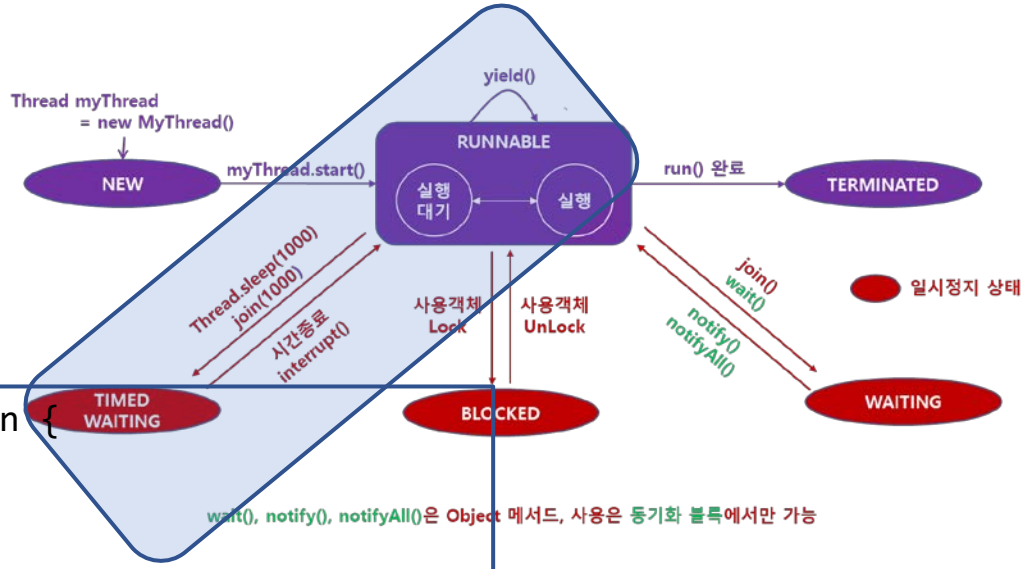
2



계속

Thread 상태(State) (TIMED_WAITING)

join() – interrupt()




```
public static void main(String[] ar) throws InterruptedException {

  //#1. join(...) - interrupt() Test
  MyThread1 myThread1 = new MyThread1();
1 MyThread2 myThread2 = new MyThread2(myThread1);
  myThread1.start();
  myThread2.start();

  Thread.sleep(100); //시간지연
2 System.out.println("MyThread1 State : " + myThread1.getState()); //RUNNABLE
  System.out.println("MyThread2 State : " + myThread2.getState()); //TIMED_WAITING

  myThread2.interrupt();
3 Thread.sleep(100);
  System.out.println("MyThread1 State : " + myThread1.getState()); //RUNNABLE
  System.out.println("MyThread2 State : " + myThread2.getState()); //RUNNABLE

}
```

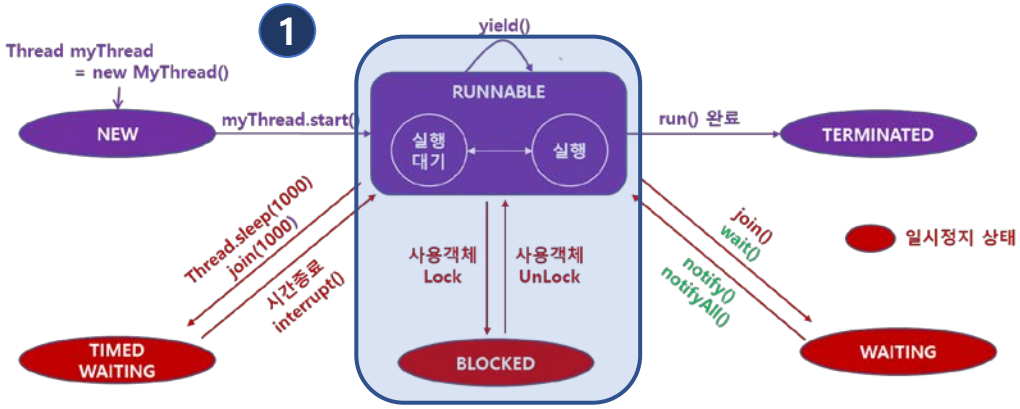
```
MyThread1 State : RUNNABLE
MyThread2 State : TIMED_WAITING
 --join() 중 interrupt 발생--
MyThread1 State : RUNNABLE
MyThread2 State : RUNNABLE
```

The End

Thread 상태(State) #2

Thread 상태(State)

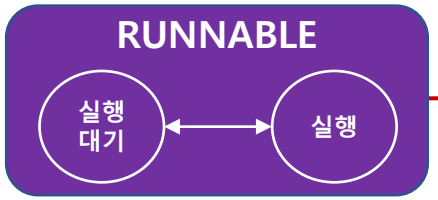
EJB 내부적으로 thread로 작동하므로 이해하는 데 도움됨



☞ 쓰레드의 상태 (**BLOCKED**)

2

동기화 메서드 또는
동기화 블록으로 **Lock이 걸린 경우**



사용객체 Lock

BLOCKED

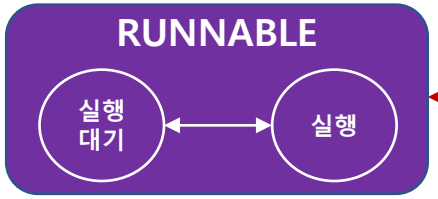
3

일정시간 일시정지상태를 나타내며
이 상태에서는 CPU를 사용할 수 없음

사용객체의 Lock이 풀리고 사용을 위한
Key를 획득하면 Runnable 상태로 전환

4

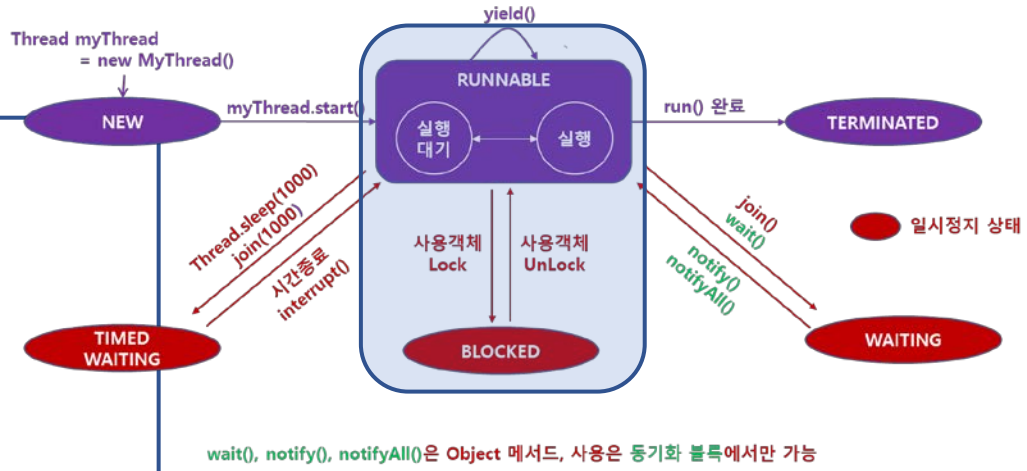
동기화 메서드 또는 동기화 블록
실행 완료로 **Lock이 풀린 경우**



사용객체 UnLock

BLOCKED

Thread 상태(State) (BLOCKED)



wait(), notify(), notifyAll()은 Object 메서드, 사용은 동기화 블록에서만 가능

```
class MyBlockTest {

  MyClass mc = new MyClass(); //공유객체

  Thread t1 = new Thread("thread1") {
    public void run() { mc.syncMethod(); }
  };
  Thread t2 = new Thread("thread2") {
    public void run() { mc.syncMethod(); }
  };
  Thread t3 = new Thread("thread3") {
    public void run() { mc.syncMethod(); }
  };
  void startAll() { //세개의 쓰레드 모두 시작
    t1.start();     t2.start();     t3.start();
  }

  class MyClass{
    synchronized void syncMethod() {
      try {Thread.sleep(100);} catch (InterruptedException e) {}
      System.out.println("["+Thread.currentThread().getName()+"]");
      System.out.println("thread1->"+t1.getState());
      System.out.println("thread2->"+t2.getState());
      System.out.println("thread3->"+t3.getState());
      for(long i=0; i<100000000L; i++) {}
    }
  }
}
```

1 3 4 2

5

```
public static void main(String[] args) {

  MyBlockTest mbt = new MyBlockTest();
  mbt.startAll();

}
```

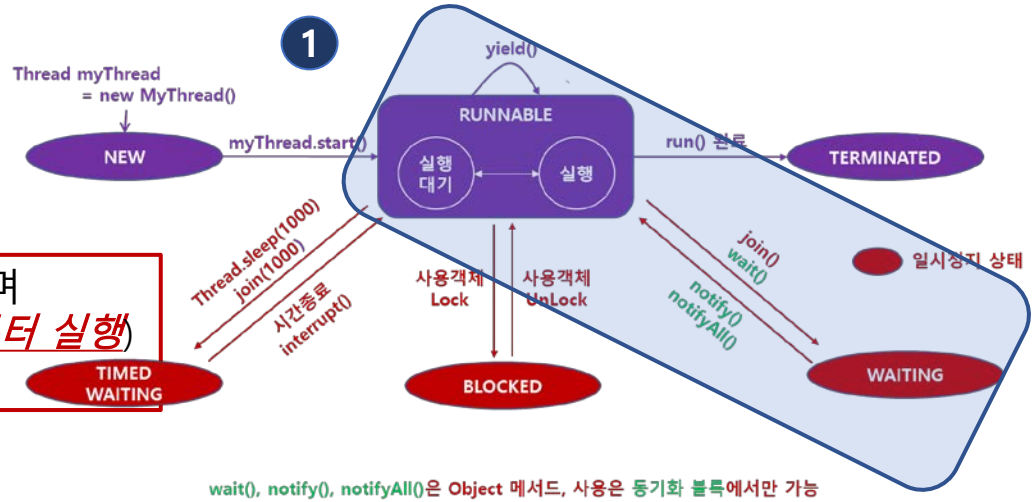
6

```
[thread1]
thread1->RUNNABLE
thread2->BLOCKED
thread3->BLOCKED
[thread3]
thread1->TERMINATED
thread2->BLOCKED
thread3->RUNNABLE
[thread2]
thread1->TERMINATED
thread2->RUNNABLE
thread3->TERMINATED
```

thread는 순차접근 X
어느 thread가 될지 아무도 모름

Thread 상태(State)

☞ 쓰레드의 상태 (**WAITING**)



2 **Object 메서드**. wait()를 호출한 메서드는 일시정지 상태가 되며 다른 쓰레드가 notify(), notifyAll()로만 깨울 수 있음 (*wait() 다음줄부터 실행*)
동기화메서드, 동기화블록내에서만 사용 가능

`void wait()`

RUNNABLE (실행 대기 / 실행)

wait()
join()

WAITING

`void join()`

3 **thread1.join()**과 같이 join시 시간을 주지 않으면 thread1이 종료할때 까지 **호출한 쓰레드는 일시정지** 상태

join()의 경우 실행이 완료되거나 interrupt() 메서드 호출로 RUNNABLE 상태로 전환
wait()는 다른 쓰레드에서 notify()/notifyAll()을 호출해야 다시 RUNNABLE 상태로 전환

RUNNABLE (실행 대기 / 실행)

실행완료 또는 interrupt()
notify(), notifyAll()

WAITING

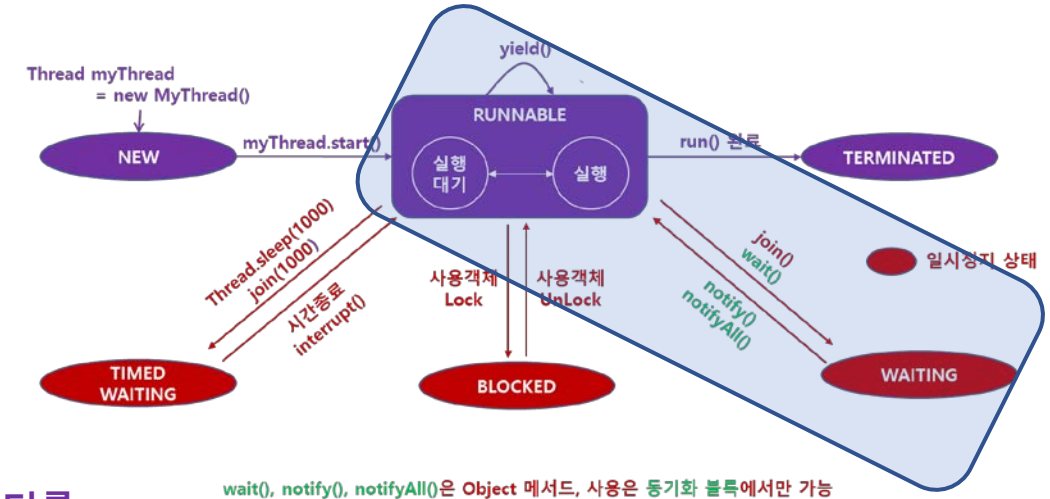
4 `void notify(), void notifyAll()`

Object 메서드. notify()는 하나의 쓰레드를, notifyAll()는 wait() 중인 모든 Thread를 깨움
동기화메서드, 동기화블록 내에서만 사용 가능

Thread 상태(State)

☞ 쓰레드의 상태 (**WAITING**)

- 적용예시



1

데이터 하나를
저장할 수 있는 객체

데이터를 하나 꺼내
객체에 저장하는 역할

SetThread

data

객체의 데이터를
하나씩 꺼내 읽는 역할

GetThread

data1 data2 data3 data4
data5 data6 data7 data8

data8
⋮
data2
data1

2

Without wait()/notify()

데이터를 쓰는 속도
≠ 데이터를 읽는 속도

data5
⋮
data3
null

3

With wait()/notify()

데이터쓰기 → 데이터읽기
의 순으로 진행

wait()-notify(), notifyAll()

data8
⋮
data2
data1

Thread 상태(State) (WAITING - WithoutWaitNotify)

1

```
class DataBox {

  int data;
  synchronized void inputData(int data)
                  throws InterruptedException {
    this.data = data;
    System.out.println("입력 데이터: "+data);
  }

  synchronized void outputData()
                  throws InterruptedException {
    System.out.println("출력 데이터: "+data);
  }

}
```

2

```
public static void main(String[] args) {
  DataBox dataBox = new DataBox();
  Thread t1 = new Thread() { //쓰기 쓰레드
  @Override
    public void run() {
      for (int i = 0; i < 10; i++) {
        try {
          dataBox.inputData(i);
        } catch (InterruptedException e) {}
      }
    }
  };

  Thread t2 = new Thread() { //읽기 쓰레드
    @Override
    public void run() {
      for (int i = 0; i < 10; i++) {
        try {
          dataBox.outputData();
        } catch (InterruptedException e) {}
      }
    }
  };

  t1.start();    t2.start();
}
```

3

4

5

```
입력 데이터: 0
입력 데이터: 1
입력 데이터: 2
출력 데이터: 2
출력 데이터: 2
출력 데이터: 2
출력 데이터: 2
출력 데이터: 2
출력 데이터: 2
출력 데이터: 2
출력 데이터: 2
출력 데이터: 2
출력 데이터: 2
출력 데이터: 2
입력 데이터: 3
입력 데이터: 4
입력 데이터: 5
입력 데이터: 6
입력 데이터: 7
입력 데이터: 8
입력 데이터: 9
```

Thread 상태(State) (WAITING - WithWaitNotify)

1

2

Step 1. 쓰기 쓰레드 동작 (데이터출력)
Step 2. 읽기 쓰레드 깨우기 (notify())
Step 3. 쓰기 쓰레드 일시정지 (wait())

Step 4. 읽기 쓰레드 동작 (데이터읽기)
Step 5. 쓰기 쓰레드 깨우기 (notify())
Step 6. 읽기 쓰레드 일시정지 (wait())
Step 1~6. 반복

```
class DataBox {

  boolean isEmpty=true;
  int data;
  synchronized void inputData(int data)
                    throws InterruptedException {
    if(!isEmpty) {
      wait();
    }
    isEmpty=false;
    this.data = data;
    System.out.println("입력 데이터: "+data);
    notify();
  }

  synchronized void outputData()
                    throws InterruptedException {
    if(isEmpty) {
      wait();
    }
    isEmpty=true;
    System.out.println("출력 데이터: "+data);
    notify();
  }
}
```

3

4

Thread 상태(State) (WAITING - WithWaitNotify)

```
public static void main(String[] args) {
  DataBox dataBox = new DataBox();

  Thread t1 = new Thread() { //쓰기 쓰레드
  @Override
    public void run() {
      for (int i = 0; i < 10; i++) {
        try {
          dataBox.inputData(i);
        } catch (InterruptedException e) {}
      }
    }
  };

  Thread t2 = new Thread() { //읽기 쓰레드
    @Override
    public void run() {
      for (int i = 0; i < 10; i++) {
        try {
          dataBox.outputData();
        } catch (InterruptedException e) {}
      }
    }
  };

  t1.start();  t2.start();
}
```

1

2

```
입력 데이터: 0
출력 데이터: 0
입력 데이터: 1
출력 데이터: 1
입력 데이터: 2
출력 데이터: 2
입력 데이터: 3
출력 데이터: 3
입력 데이터: 4
출력 데이터: 4
입력 데이터: 5
출력 데이터: 5
입력 데이터: 6
출력 데이터: 6
입력 데이터: 7
출력 데이터: 7
입력 데이터: 8
출력 데이터: 8
입력 데이터: 9
출력 데이터: 9
```

The End