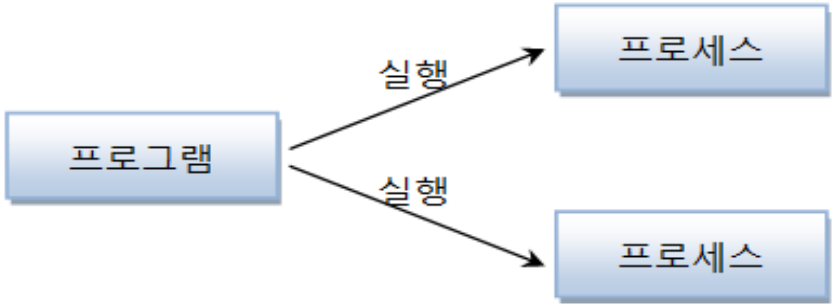
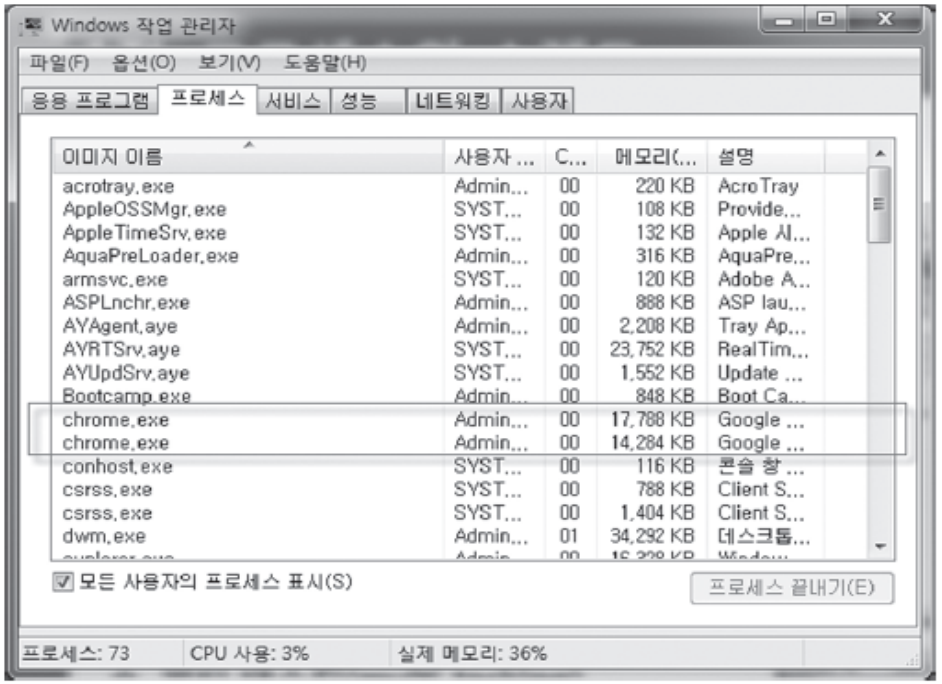
스레드와 프로세스

멀티 스레드

❖ 프로세스(process)

- 실행 중인 하나의 프로그램
- 하나의 프로그램이 여러 프로세스로 만들어짐

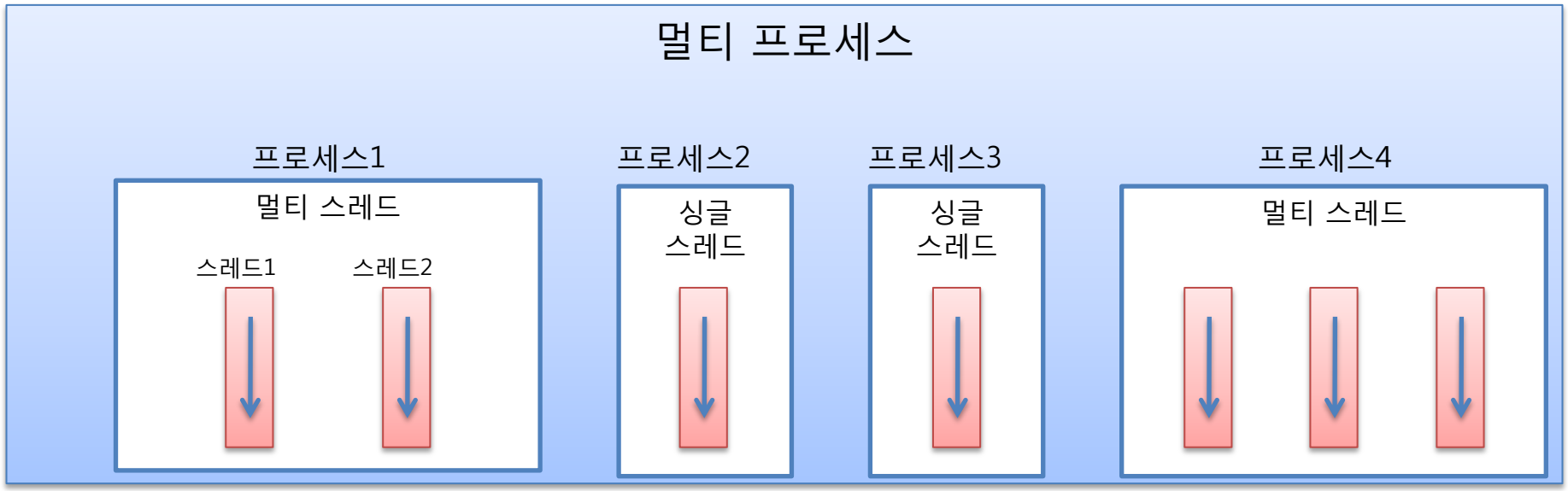




멀티 스레드

❖ 멀티 태스킹(multi tasking)

- 두 가지 이상의 작업을 동시에 처리하는 것
- 멀티 프로세스
  - 독립적으로 프로그램들을 실행하고 여러 가지 작업 처리
- 멀티 스레드
  - 한 개의 프로그램을 실행하고 내부적으로 여러 가지 작업 처리



Single Thread VS Multi Thread

❖ 스레드의 특징

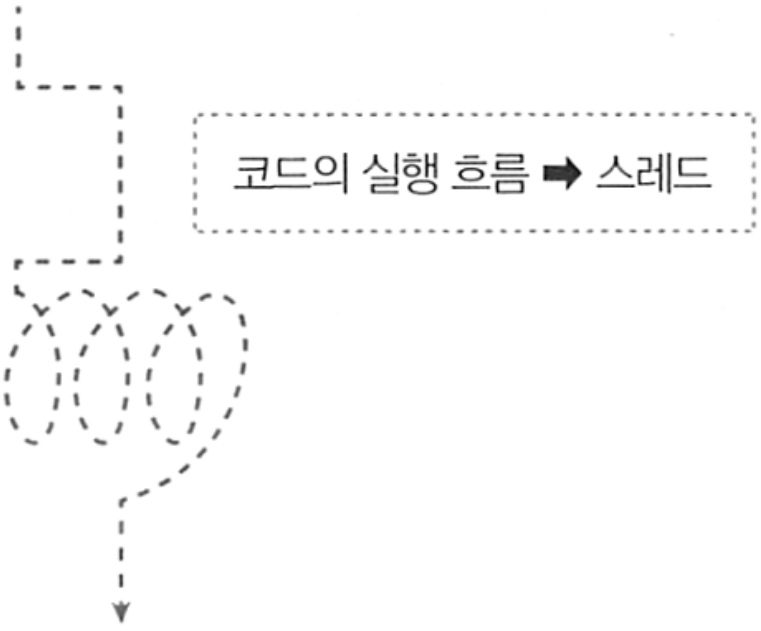
- ㅇ 프로세스
  - ‘액티브 프로그램’의 의미로 해석
  - 하나의 프로세스는 자신만의 전용 공간과 자원을 할당 받은 상태에서 독점적으로 일을 처리하는 구조
- ㅇ 스레드
  - 프로세스의 내부에 존재하며 공간과 자원을 공유
  - 경량 프로세스

멀티 스레드

❖ 메인(main) 스레드

- 모든 자바 프로그램은 메인 스레드가 main() 메소드 실행하며 시작
- main() 메소드의 첫 코드부터 아래로 순차적으로 실행

```
public static void main(String[] args) {
    String data = null;
    if(…) {
    }
    while(...) {
    }
    System.out.println("…");
}
```



멀티 스레드

❖ 메인(main) 스레드

- 실행 종료 조건
  - 마지막 코드 실행
  - return 문을 만나면

- main 스레드는 작업 스레드들을 만들어 병렬로 코드들 실행
  - 멀티 스레드 생성해 멀티 태스킹 수행

- 프로세스의 종료
  - 싱글 스레드: 메인 스레드가 종료하면 프로세스도 종료
  - 멀티 스레드: 실행 중인 스레드가 하나라도 있다면, 프로세스 미종료

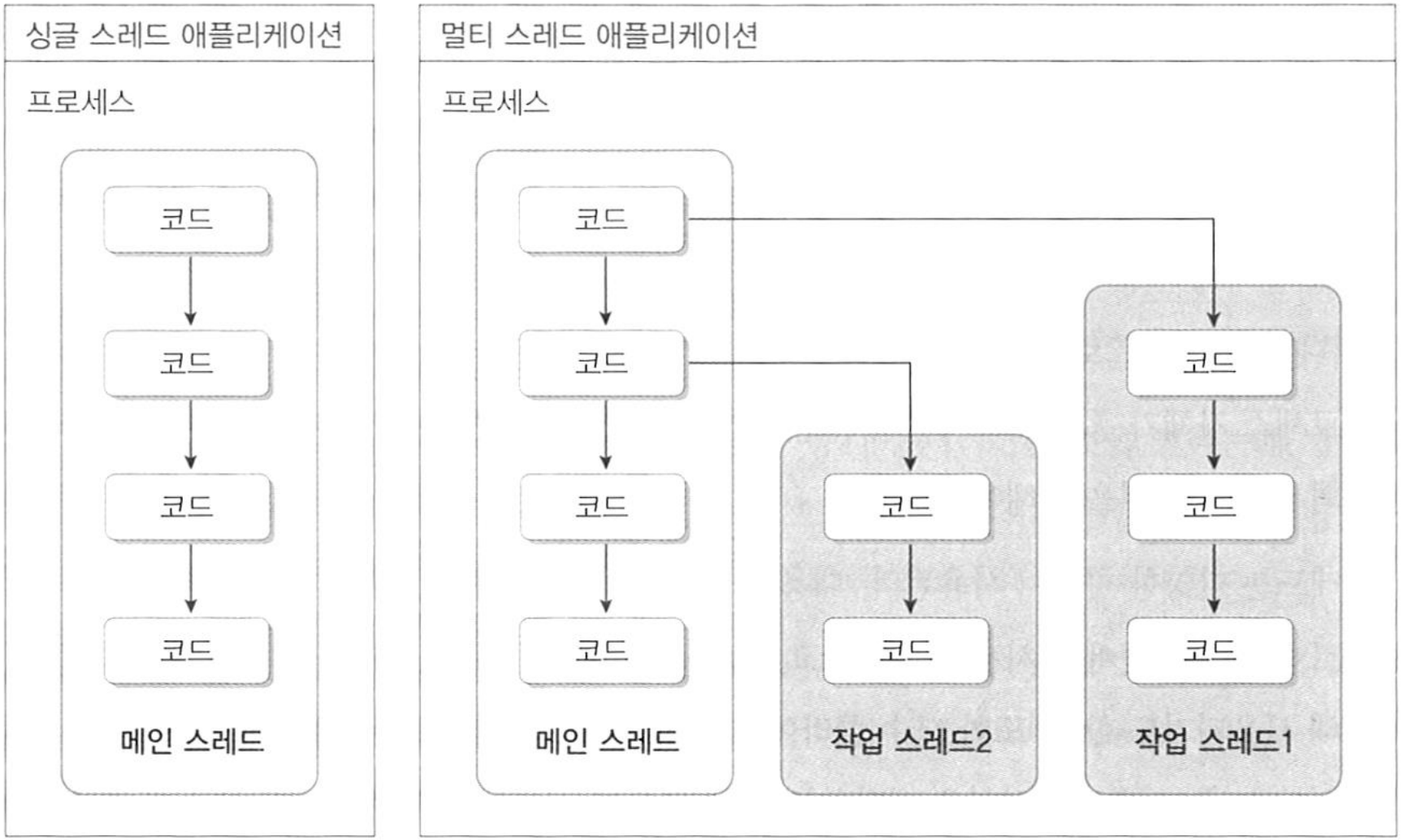
Single Thread VS Multi Thread

❖ Multi Thread 프로그램

- 여러 개의 스레드가 동시에 동작하는 프로그램
- 대표적 예
  - GUI를 가지는 프로그램
    - 창에서 발생하는 이벤트를 처리하는 부분
    - 마우스의 움직임에 따른 이벤트를 처리하는 부분 등
    - 여러 가지 스레드가 동시에 수행
  - 채팅 프로그램
    - 하나의 서버에 여러 대의 클라이언트가 접속하는 형태

멀티 스레드

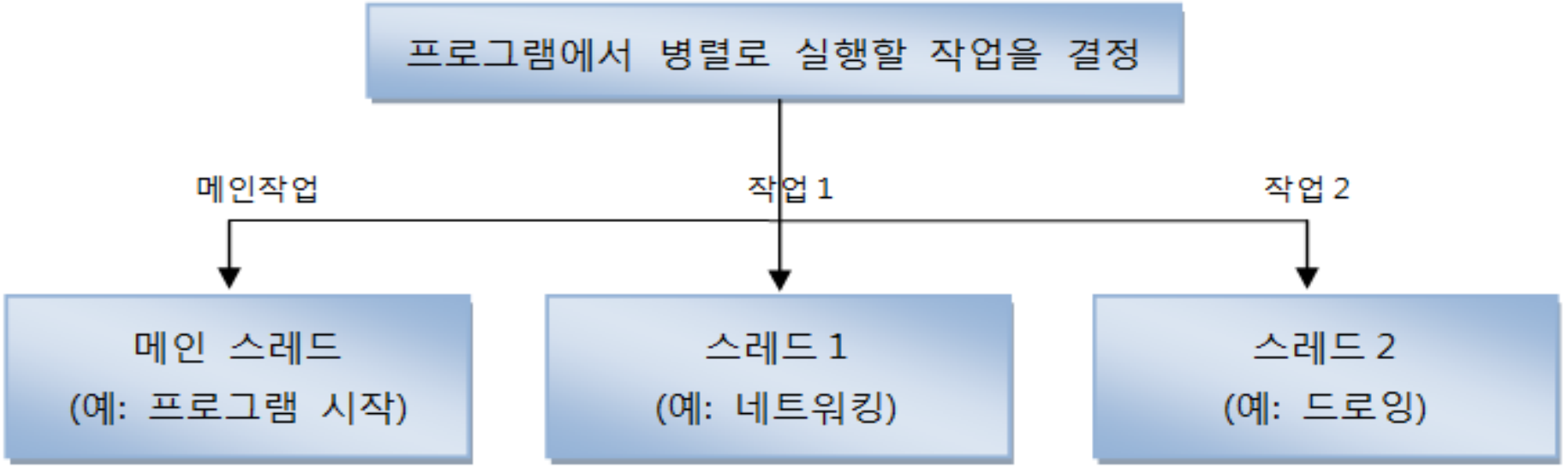
❖ 멀티 스레드


싱글 스레드 애플리케이션
프로세스
코드
코드
코드
코드
메인 스레드
멀티 스레드 애플리케이션
프로세스
코드
코드
코드
코드
메인 스레드
코드
코드
작업 스레드2
코드
코드
코드
작업 스레드1

멀티 스레드

❖ 멀티 스레드로 실행하는 어플리케이션 개발

- ㅇ 몇 개의 작업을 병렬로 실행할지 결정하는 것이 선행되어야



Single Thread VS Multi Thread

❖ Single Thread 프로그램

- 프로그램의 흐름이 하나의 큰 줄기밖에 없는 프로그램

```
public class SingleThread {
   public static void main(String[] args) {
      System.out.println("싱글 Thread 프로그램이다.");
      for (int i = 0; i < 100; i++) {
         System.out.println(Thread.currentThread().getName() + i);
      }
   }

}
```

Single Thread VS Multi Thread

❖ 스레드 만들기 1

- Thread 클래스 상속

```
public class FirstThread extends Thread {
    ……
    public void run() {
            …
    }
}
```

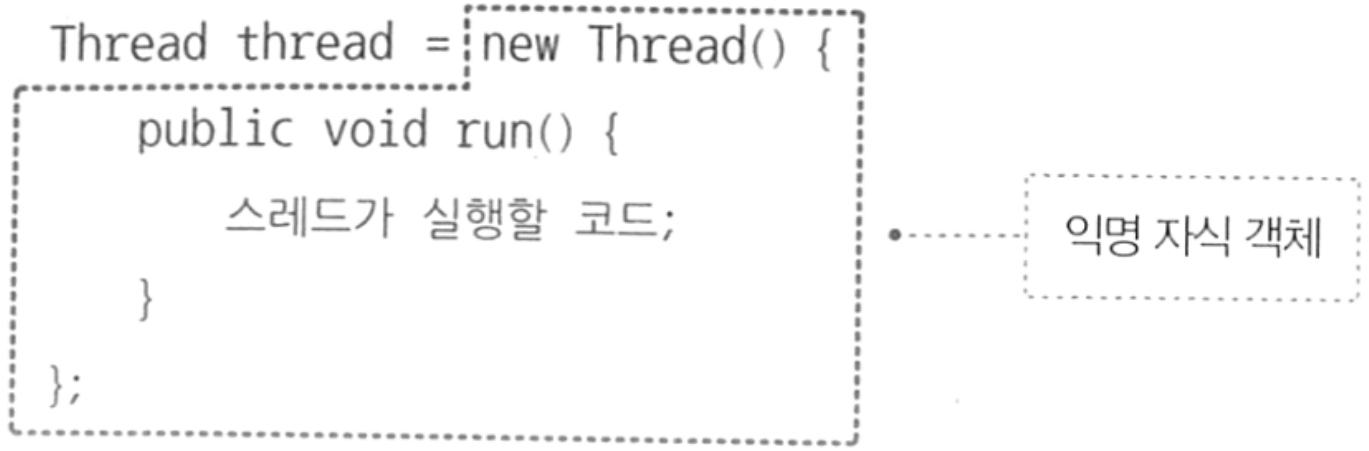
- 스레드 실행

```
FirstThread ft = new FirstThread();
ft.start();   // 스레드 시작
```

Single Thread VS Multi Thread

❖ 스레드 만들기 1

- Thread 클래스 상속 (익명 객체)



Single Thread VS Multi Thread

❖ 스레드 만들기 2

- Runnable 인터페이스 구현
  - 이미 다른 클래스를 상속한 클래스를 스레드로 운영하려는 경우

```
public class SecondThread implements Runnable {
    ……
    public void run() {
          …
    }
}
```

```
Thread thread = new Thread(st);
thread.start();  // 스레드 시작
```

멀티 스레드

❖ 작업 스레드 생성 방법 2

- Runnable 인터페이스의 익명 구현 객체 사용

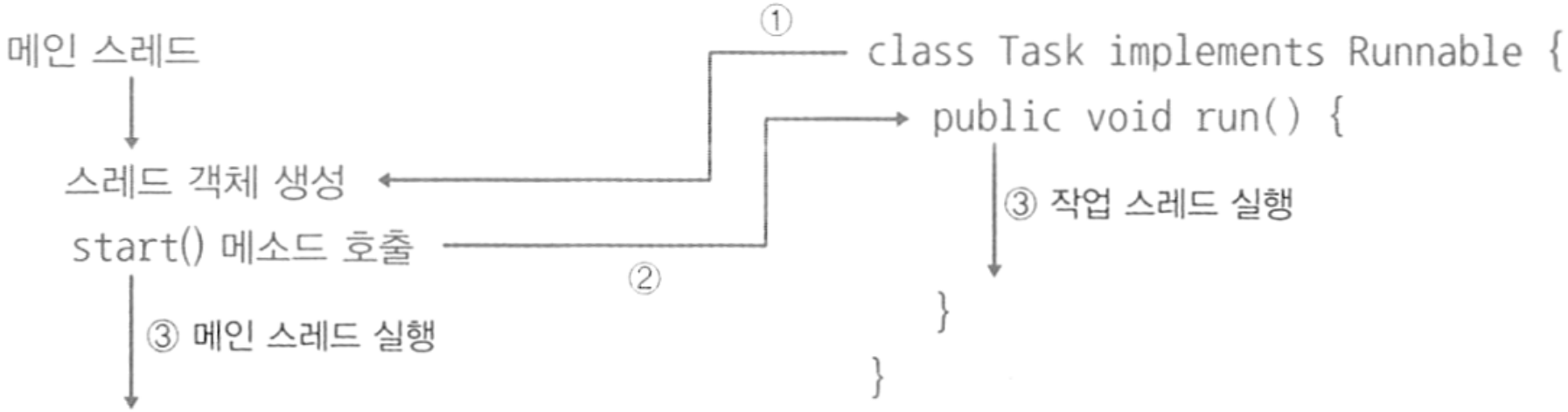
```
Thread thread = new Thread(new Runnable() {
    public void run() {
        스레드가 실행할 코드;
    }
});
```

익명 구현 객체

멀티 스레드

❖ 작업 스레드 생성 방법

- ㅇ 스레드의 실행
  - `thread.start( );`



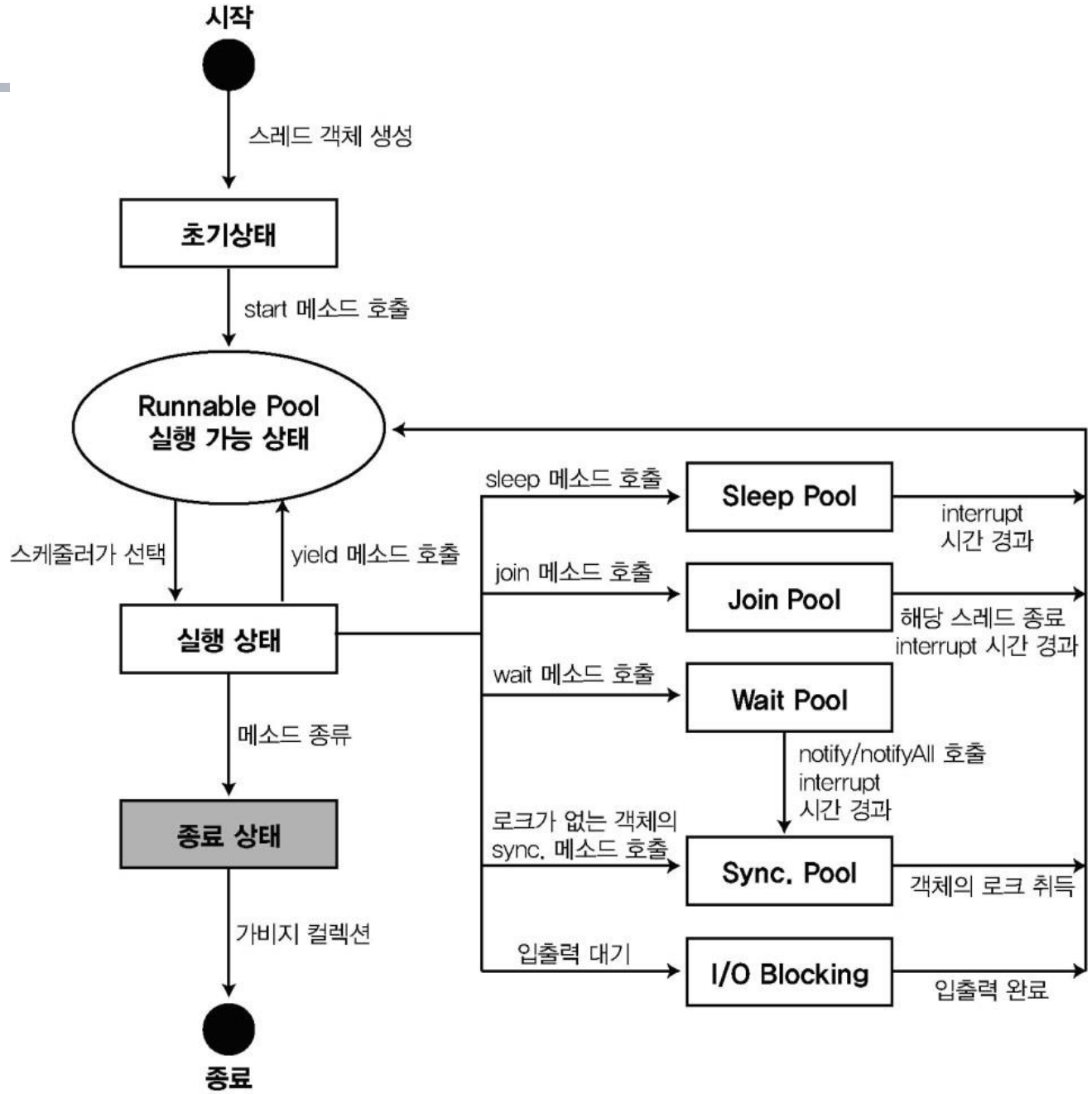
스레드의 생명주기

❖ 스레드의 상태

- alive
  - 실행 가능 상태 : Runnable pool
  - 실행 상태 : CPU 배정
  - 대기 상태 : 특정 조건을 만족할 때까지 대기
    - sleep pool,
    - wait pool,
    - join pool,
    - I/O blocking pool
- dead
  - 스레드의 run() 메소드를 완전히 수행하여 더 수행할 코드가 남아 있지 않은 경우
  - stop() 메소드에 의하여 종료되는 경우

스레드의 생명주기

❖ 스레드의 실행 상태도



스레드의 생명주기와 관련된 메소드

❖ sleep() 메소드

- 현재 실행중인 스레드를 주어진 시간(밀리세컨드) 동안 Sleep Pool로 전이
- 시간이 모두 경과되거나 사용자가 interrupt를 호출하는 경우
  - Sleep Pool에서 빠져나올 수 있으며 빠져나와 실행 가능 상태(Runnable Pool)로 전이

❖ join() 메소드

- 다른 스레드와 협동 작업을 요구할 때 사용
- 지정한 스레드가 종료할 때까지 Join Pool에서 대기
- join() 메시지를 전달받은 스레드가 종료 상태가 되면
  - Join Pool에서 대기 중이었던 스레드가 풀려 나와서 실행 가능 상태가 되어 Runnable Pool로 전이
- timeout 시간 지정 가능

스레드의 생명주기와 관련된 메소드

❖ synchronized 메소드 호출

- 여러 개의 스레드가 하나의 공유 자원 객체를 접근할 때
  - 공유 자원 객체에 존재하고 있는 필드를 보호
- 이 키워드가 적용된 메소드를 호출하는 경우
  - 공유 객체에 존재하는 모니터링 로크를 메소드를 호출한 스레드가 획득하고, 객체를 잠금
- 다른 스레드가 객체의 synchronized 메소드를 호출한 경우
  - 메소드를 실행하지 못하고 모니터링 로크가 반환되어 객체의 잠금 상태가 풀릴 때까지 대기해야

→ synchronized blocking

스레드의 생명주기와 관련된 메소드

❖ wait(), notify(), notifyAll() 메소드

- Object의 메서드
  - 모든 객체가 상속 받음
  - synchronized 키워드와 밀접한 관련
    - 공유 객체에 있는 모니터링 로크의 획득, 반환 메커니즘과도 연결되어 있음
- wait() 메소드
  - 스레드를 기다리게 만드는 메소드
  - 모든 객체는 Wait Pool을 가지고 있음
    - wait() 메소드가 호출되면 현재 실행 중인 스레드는 wait() 메시지를 전달받은 객체의 Wait Pool로 들어가며 대기 상태가 됨
- notify() 메서드
  - Wait Pool에 여러 개의 스레드가 존재할 경우 그 중 하나의 스레드만 빠져나와 실행 가능 상태가 되어 Runnable Pool로 전이
- notifyAll()
  - Wait Pool에 있는 모든 스레드를 Runnable Pool로 전이

스레드의 생명주기와 관련된 메소드

❖ 스레드의 종료

- 정상적인 종료
  - run() 메소드의 내용을 모두 수행하면 스레드는 자동으로 종료
- 비정상적인 종료
  - run() 메소드를 모두 수행하기 전에 스레드를 종료
  - 안전하게 종료하기 위해서 논리형 객체를 이용해서 원하는 상황이 되면 run() 메소드를 빠져나가도록 코드를 작성

디자인패턴1 (Single Threaded Execution Pattern)

Single Threaded Execution Pattern

❖ **패턴을 적용하지 않은 프로그램**

- 다섯 명의 기사 : 각각은 스레드
- 한 명의 기사만 건너갈 수 있는 다리 : 고유 객체
- 다리를 건널 때 자신의 신원을 기록하고 건너 감

Single Threaded Execution Pattern

❖ 패턴을 적용하지 않은 프로그램

- Bridge.java

```
public class Bridge {
   private int counter = 0;
   private String name = "아무개";
   private String address = "모름";

   public void across(String name, String address) {
      this.counter++;
      this.name = name;
      this.address = address;
      check();
   }

   public String toString() {
      return   "이름:" + name + ", 출신:" + address +
               ", 도전 횟수:" + counter;
   }

   public void check() {
      if (name.charAt(0) != address.charAt(0)) {
         System.out.println("문제 발생!!!! " + toString());
      }
   }
}
```

Single Threaded Execution Pattern

❖ 패턴을 적용하지 않은 프로그램

○ Knight.java

```
public class Knight extends Thread{
   private final Bridge bridge; // 공유 객체

   private final String name;
   private final String address;

   public Knight(Bridge bridge, String name, String address) {
      this.bridge = bridge;
      this.name = name;
      this.address = address;
   }

   public void run() {
      System.out.println(name + " 기사가 도전한다.");
      while (true) {
         bridge.across(name, address);
      }
   }
}
```

Single Threaded Execution Pattern

❖ 패턴을 적용하지 않은 프로그램

- BridgeTest.java

```
public class BridgeTest {
   public static void main(String[] args) {
      System.out.println("시뮬레이션을 시작한다.");
      Bridge bridge = new Bridge();

      new Knight(bridge, "홍길동", "홍천").start();
      new Knight(bridge, "임꺽정", "임실").start();
      new Knight(bridge, "일지매", "일산").start();
      new Knight(bridge, "장보고", "장흥").start();
      new Knight(bridge, "이순신", "이천").start();
   }

}
```

Single Threaded Execution Pattern

❖ 패턴을 적용하지 않은 프로그램

- 실행결과

```
시뮬레이션을 시작한다.
홍길동 기사가 도전한다.
임꺽정 기사가 도전한다.
일지매 기사가 도전한다.
장보고 기사가 도전한다.
이순신 기사가 도전한다.
문제 발생!!!! 이름:홍길동, 출신:홍천, 도전 횟수:2836450
문제 발생!!!! 이름:임꺽정, 출신:임실, 도전 횟수:3496737
문제 발생!!!! 이름:일지매, 출신:일산, 도전 횟수:4177211
문제 발생!!!! 이름:임꺽정, 출신:임실, 도전 횟수:6866630
문제 발생!!!! 이름:일지매, 출신:일산, 도전 횟수:7548705
문제 발생!!!! 이름:장보고, 출신:장흥, 도전 횟수:8183682
문제 발생!!!! 이름:이순신, 출신:이천, 도전 횟수:8854972
문제 발생!!!! 이름:장보고, 출신:장흥, 도전 횟수:11563128
문제 발생!!!! 이름:홍길동, 출신:임실, 도전 횟수:12932671
문제 발생!!!! 이름:일지매, 출신:일산, 도전 횟수:14293669
문제 발생!!!! 이름:장보고, 출신:장흥, 도전 횟수:14928019
```

Single Threaded Execution Pattern

❖ Single Threaded Execution Pattern의 적용

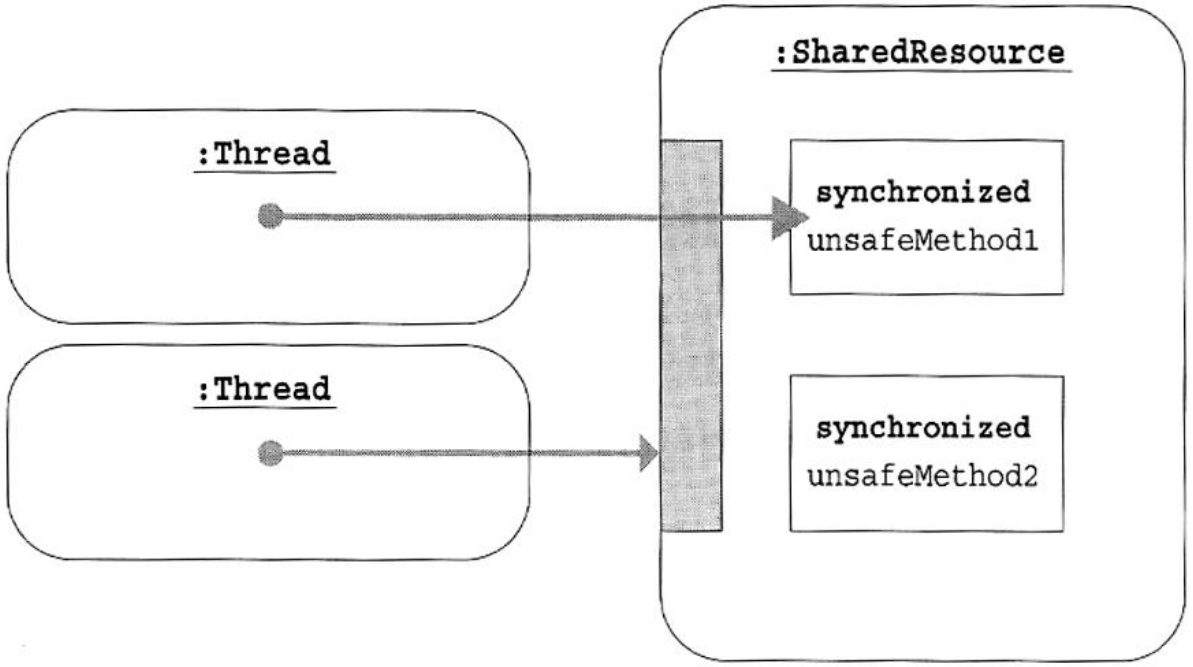
○ Bridge.java

```
public class Bridge {
   private int counter = 0;
   private String name = "아무개";
   private String address = "모름";

   public synchronized void across(String name, String address) {
      this.counter++;
      this.name = name;
      this.address = address;
      check();
   }

   public synchronized String toString() {
      return   "이름:" + name + ", 출신:" + address +
               ", 도전 횟수:" + counter;
   }

   public void check() {
      if (name.charAt(0) != address.charAt(0)) {
         System.out.println("문제 발생!!!! " + toString());
      }
   }
}
```

Single Threaded Execution Pattern

❖ synchronized 키워드의 역할

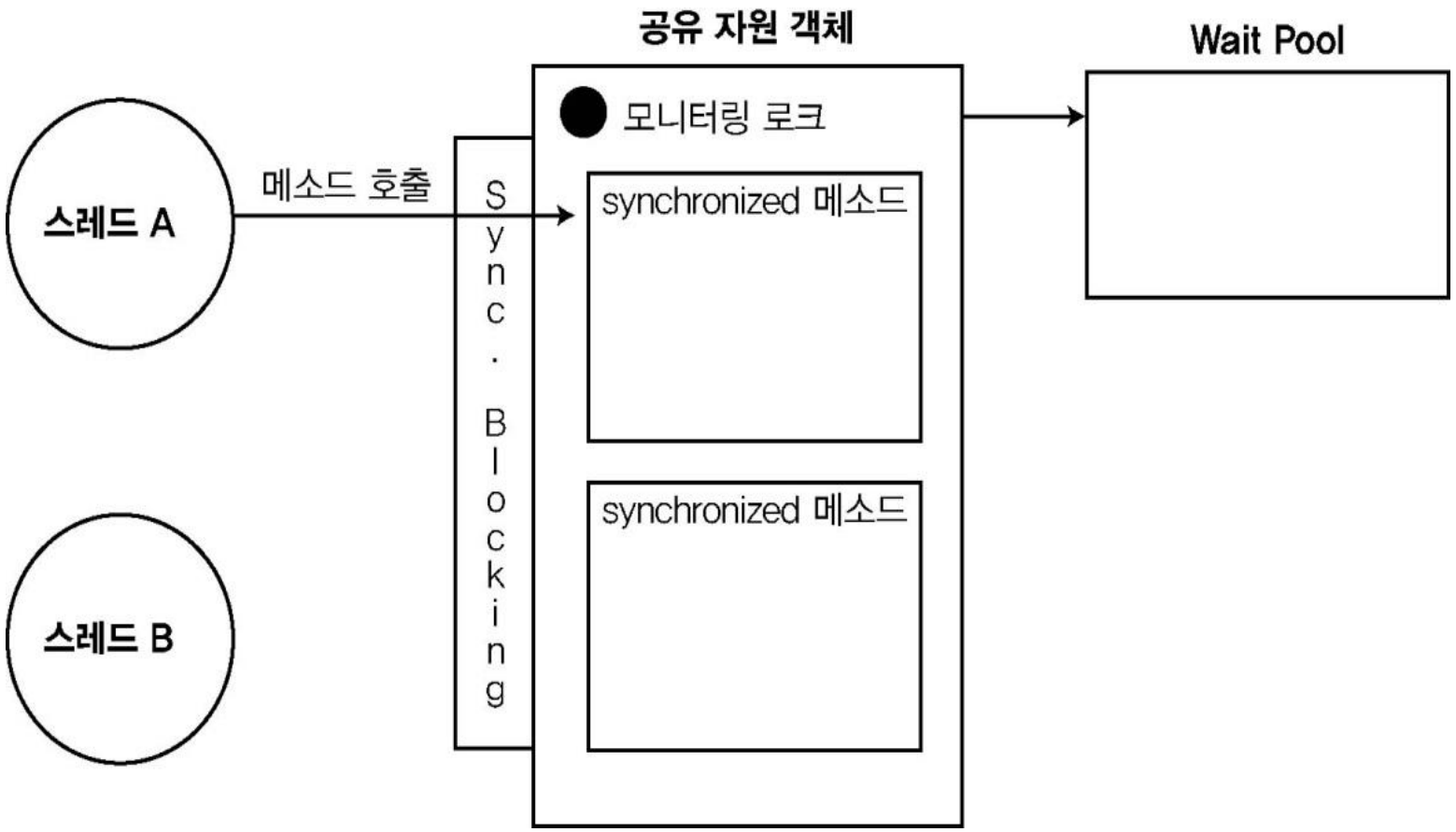
- 이 메소드를 실행하는 스레드는 해당 객체의 모니터링 로크를 얻어서 객체를 잠가 버린다.
- 잠긴 객체는 synchronized 키워드가 붙은 메소드를 실행할 수 없는 상태가 되고, across() 메소드가 종료하면서 스레드가 모니터링 로크를 객체에게 돌려주어 해제되면 그때서야 다른 스레드가 실행될 수 있다.



상호 배제 및 협력

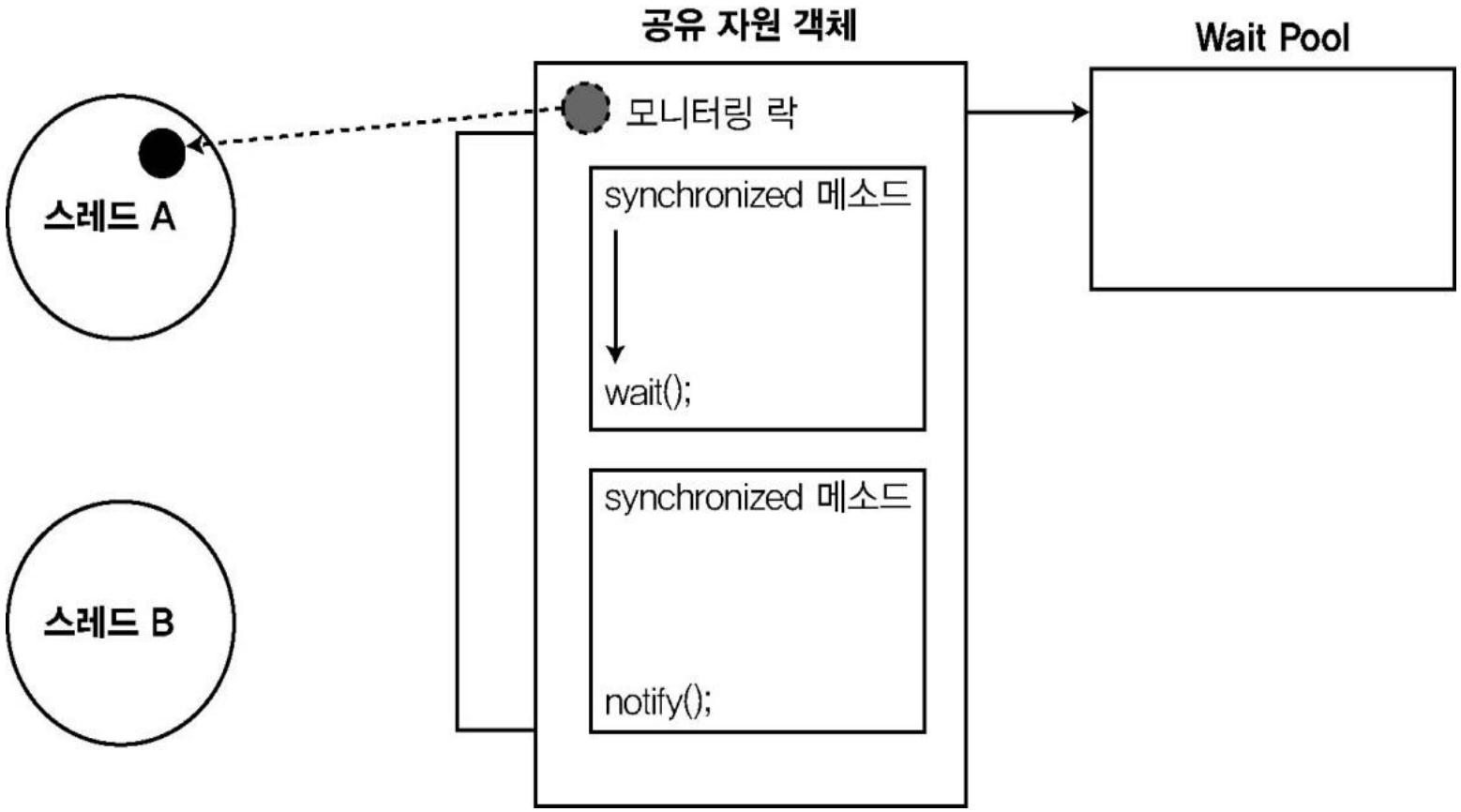
wait(), notify()

❖ 스레드A가 메소드를 호출하기 전



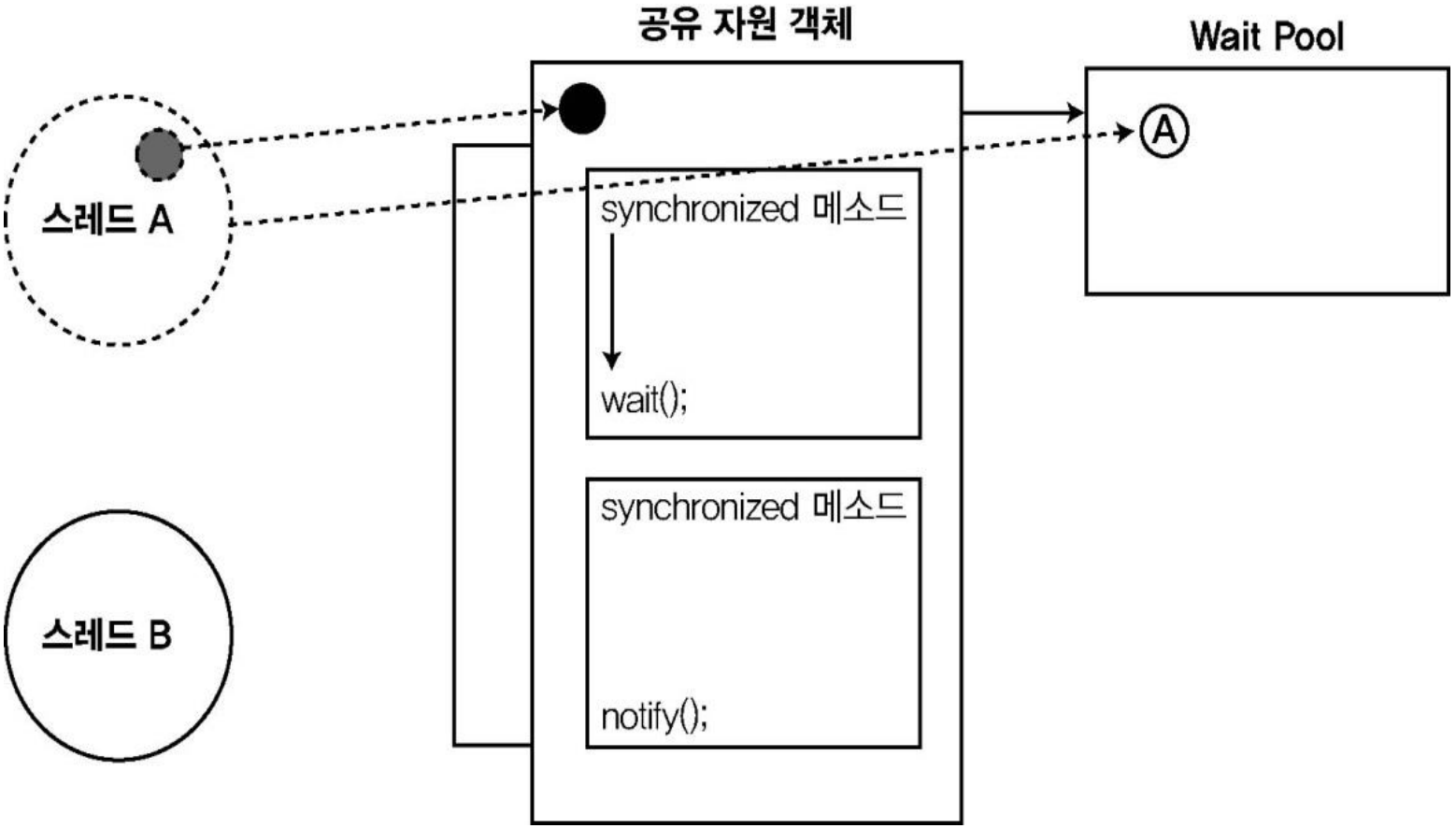
wait(), notify()

❖ 스레드A가 메소드를 실행하는 중



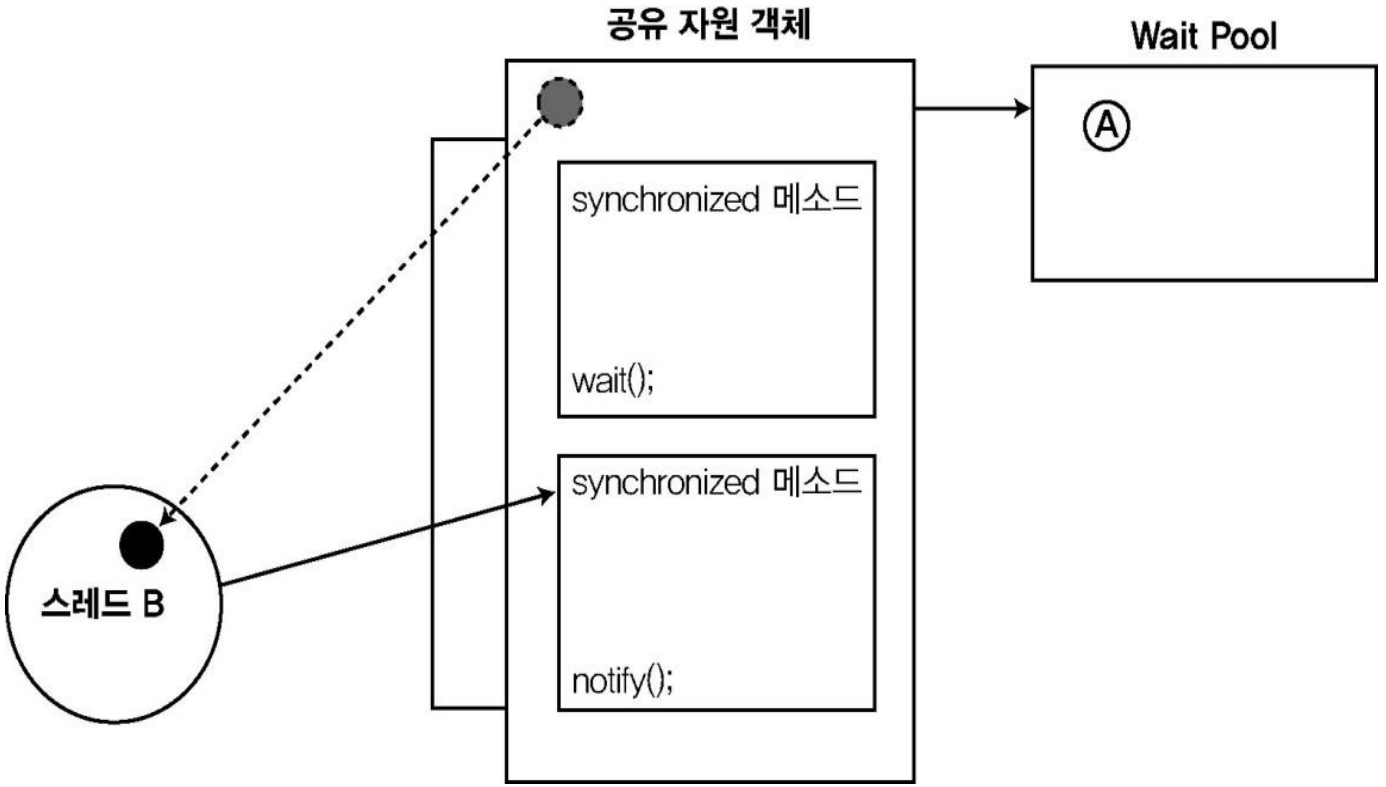
wait(), notify()

❖ 스레드 A가 wait() 실행 후



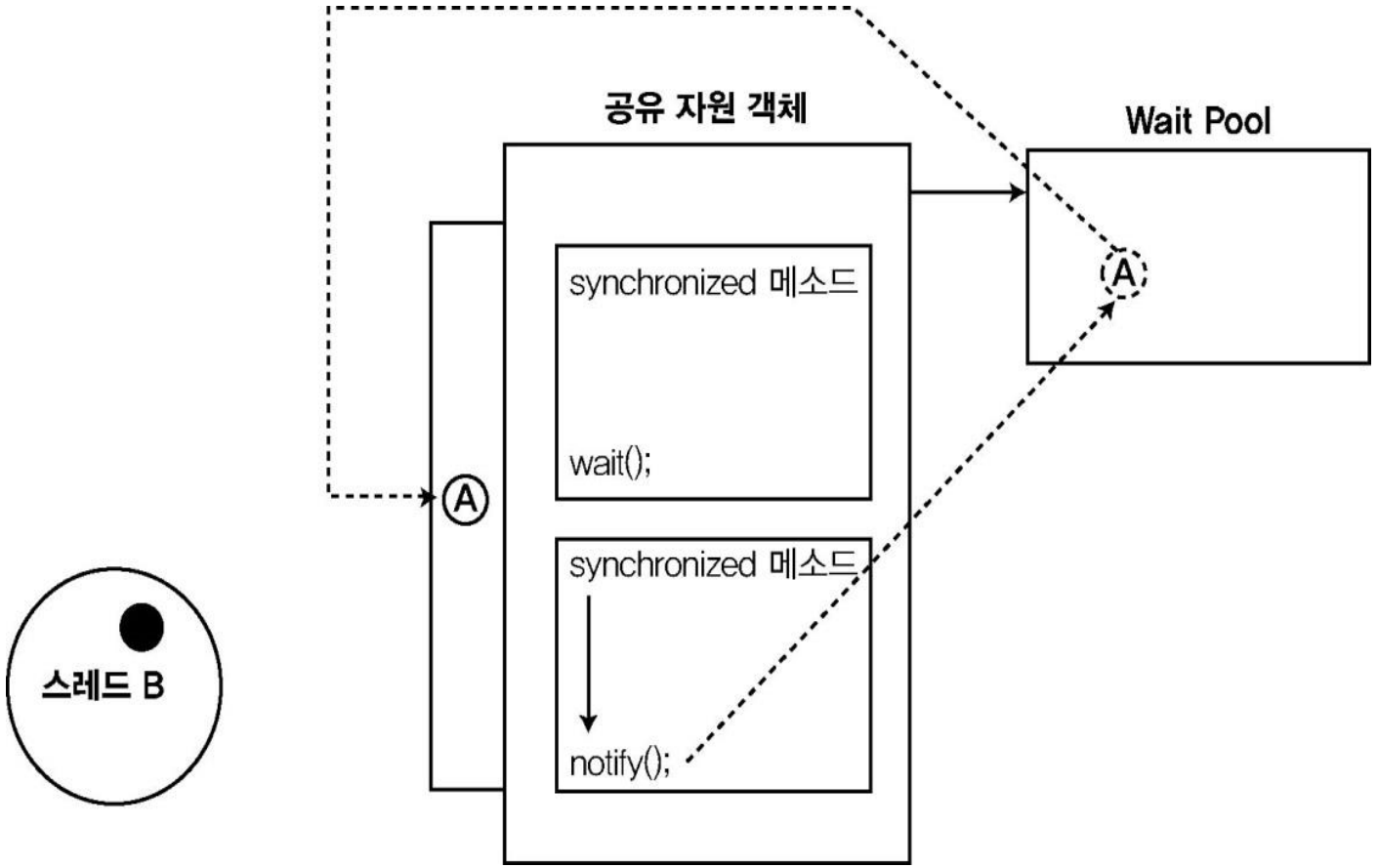
wait(), notify()

❖ 스레드 B가 메소드를 실행하는 중



wait(), notify()

❖ 스레드 B가 notify 메소드를 실행한 후



wait(), notify()

❖ 스레드 B가 메소드를 끝낸 후