

(INTERMEDIATE) JAVA PROGRAMMING

Thread and Synchronization
Chapter 13

JAVA: THREAD

Review : 중간 점검 문제



중간점검

1. 스레드와 프로세스의 결정적인 차이점은 무엇인가?
2. 스레드를 사용해야만 하는 프로그램을 생각하여 보자.
3. 멀티 스레딩에서 발생할 수 있는 문제에는 어떤 것들이 있을까? 추측하여 보라.

Review: 스레드 생성과 실행

스레드 생성 방법

Thread 클래스를 상속
하는 방법

Thread 클래스를 상속받은 후에 run()
메소드를 재정의한다.

Runnable 인터페이스를 구
현하는 방법

run() 메소드를 가지고 있는 클래스를
작성하고 ,이 클래스의 객체를 Thread
클래스의 생성자를 호출할 때 전달한
다.

Review: Thread 클래스

메소드	설명
<code>Thread()</code>	매개 변수가 없는 기본 생성자
<code>Thread(String name)</code>	이름이 name인 Thread 객체를 생성한다.
<code>Thread(Runnable target, String name)</code>	Runnable을 구현하는 객체로부터 스레드를 생성한다.
<code>static int activeCount()</code>	현재 활동 중인 스레드의 개수를 반환한다.
<code>String getName()</code>	스레드의 이름을 반환
<code>int getPriority()</code>	스레드의 우선순위를 반환
<code>void interrupt()</code>	현재의 스레드를 중단한다.
<code>boolean isInterrupted()</code>	현재의 스레드가 중단될 수 있는지를 검사
<code>void setPriority(int priority)</code>	스레드의 우선순위를 지정한다.
<code>void setName(String name)</code>	스레드의 이름을 지정한다.
<code>static void sleep(int milliseconds)</code>	현재의 스레드를 지정된 시간만큼 재운다.
<code>void run()</code>	스레드가 시작될 때 이 메소드가 호출된다. 스레드가 하여야하는 작업을 이 메소드 안에 위치시킨다.
<code>void start()</code>	스레드를 시작한다.
<code>static void yield()</code>	현재 스레드를 다른 스레드에 양보하게 만든다.

Review: 예제: sleep()

SleepTest.java

```
01 public class SleepTest {
02     public static void main(String args[]) throws InterruptedException {
03         String messages[] = { "Pride will have a fall.",
04                                "Power is dangerous unless you have humility.",
05                                "Office changes manners.",
06                                "Empty vessels make the most sound." };
07
08         for (int i = 0; i < messages.length; i++) {
09             Thread.sleep(1000);
10             System.out.println(messages[i]);
11         }
12     }
13 }
```

sleep()가 다른 메소드에 의하여 중단되면 발생하는 예외, 여기서 처리하지 않고 상위 메소드로 전달한다. 사실 여기서는 다른 메소드가 sleep()을 방해할 일이 없다.

1000밀리 초 동안 실행을 중지한다.

실행결과


```
Pride will have a fall.
Power is dangerous unless you have humility.
Office changes manners.
Empty vessels make the most sound.
```

JAVA:

THREAD: INTERRUPT

인터럽트

- 인터럽트(interrupt)는 하나의 스레드가 실행하고 있는 작업을 중지하도록 하는 메커니즘이다.

```
for (int i = 0; i < messages.length; i++) {  
    try {  
        Thread.sleep(1000);  
    } catch (InterruptedException e) {  
        // 인터럽트를 받은 것이다. 단순히 리턴한다.  
        return;  인터럽트 처리는 여기에서 해준다.  
    }  
    System.out.println(messages[i]);  
}
```

- 그런데 만약 스레드가 실행 중에 한번도 sleep()을 호출하지 않는다면 InterruptedException를 받지 못한다.

```
if (Thread.interrupted()) {  
    // 인터럽트를 받은 것이다. 단순히 리턴한다.  
    return;  
}
```


스레드 만들 때 주의 사항

- `run()` 메소드가 종료하면 스레드는 종료한다.
 - 스레드가 계속 살아있게 하려면 `run()` 메소드 내 무한루프 작성
- 한번 종료한 스레드는 다시 시작시킬 수 없다.
 - 다시 스레드 객체를 생성하고 `start()`를 호출해야 함
- 한 스레드에서 다른 스레드를 강제 종료할 수 있다.
 - 뒤에서 다룸

스레드의 상태

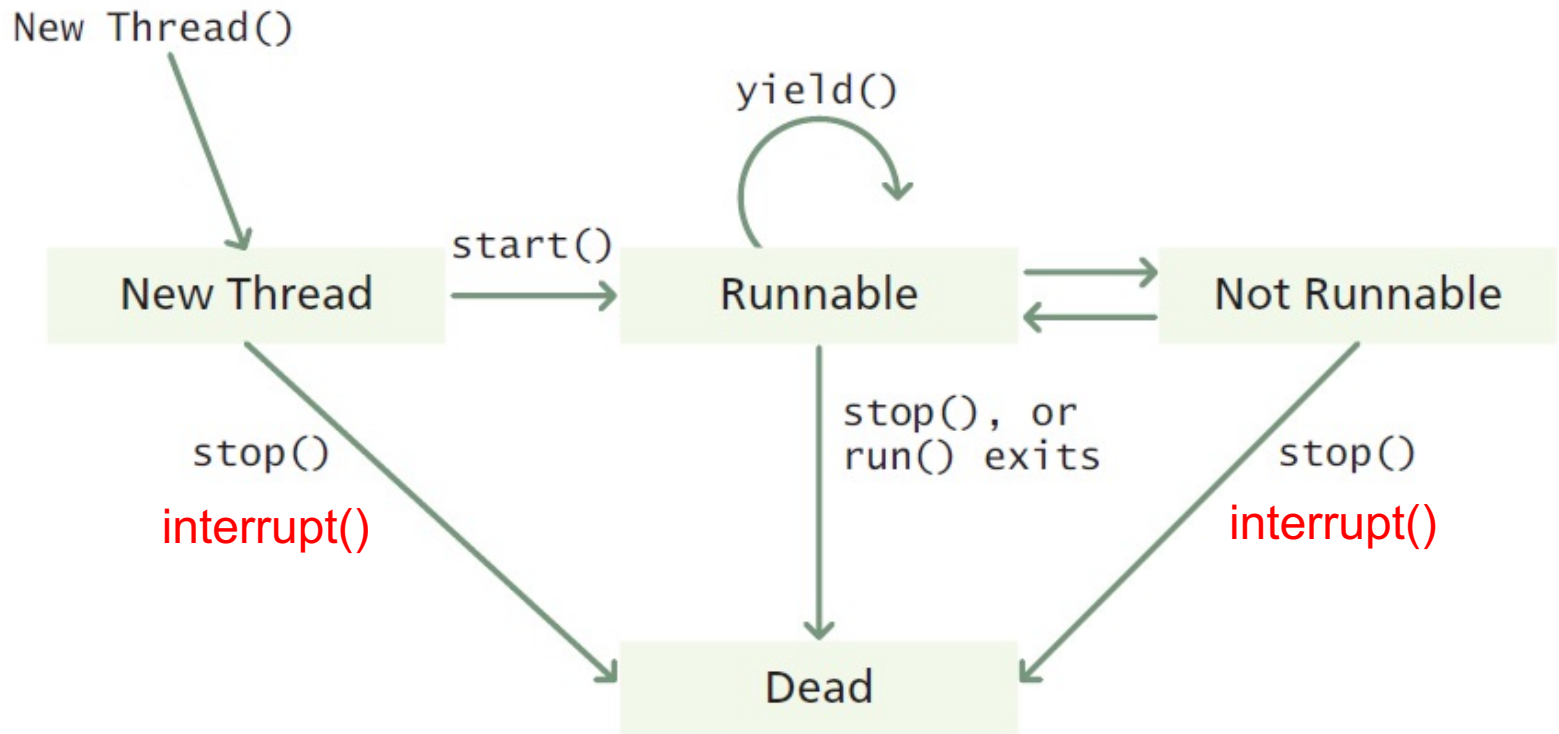


그림23-4. 스레드의 상태

JAVA: THREAD: JOIN

조인

- `join()` 메소드는 하나의 스레드가 다른 스레드의 종료를 기다리게 하는 메소드이다.

```
t.join();
```

예제

ThreadControl.java

```
01 public class ThreadControl {
02
03     static void print(String message) {
04         String threadName = Thread.currentThread().getName();
05         System.out.format("%s: %s\n", threadName, message);
06     }
07
08     private static class MessageLoop implements Runnable {
09         public void run() {
10             String messages[] = { "Pride will have a fall.",
11                                   "Power is dangerous unless you have humility.",
12                                   "Office changes manners.",
13                                   "Empty vessels make the most sound." };
14
15             try {
16                 for (int i = 0; i < messages.length; i++) {
```

메시지를 스레드 이름과
함께 출력한다.

예제

```
17         print(messages[i]);
18         Thread.sleep(2000);
19     }
20     } catch (InterruptedException e) {
21         print("아직 끝나지 않았어요!");
22     }
23 }
24 }
25
26 public static void main(String args[]) throws InterruptedException {
27     int tries = 0;
28
29     print("추가적인 스레드를 시작합니다.");
30     Thread t = new Thread(new MessageLoop());
31     t.start();
32
33     print("추가적인 스레드가 끝나기를 기다립니다.");
34     while (t.isAlive()) {
35         print("아직 기다립니다.");
36         t.join(1000);
37         tries++;
```

← 인터럽트되면 메시지를 출력한다.

← 스레드 t가 종료하기를 1초 동안 기다린다.

실행결과

```
38         if (tries > 2) {
39             print("참을 수 없네요!");
40             t.interrupt();
41             t.join();
42         }
43     }
44     print("메인 스레드 종료!");
45 }
46 }
```

스레드 t를 강제로 중단시킨다.

스레드 t가 종료하기를 기다린다.

실행결과

main: 추가적인 스레드를 시작합니다.
main: 추가적인 스레드가 끝나기를 기다립니다.
main: 아직 기다립니다.
Thread-0: Pride will have a fall.
main: 아직 기다립니다.
main: 아직 기다립니다.
Thread-0: Power is dangerous unless you have humility.
main: 참을 수 없네요!
Thread-0: 아직 끝나지 않았어요!
main: 메인 스레드 종료!

중간 점검



중간점검

1. `setPriority()`와 `getPriority()`의 역할은?
2. `sleep()` 메소드는 어떤 경우에 사용되는가?
3. 어떤 스레드가 가장 우선적으로 실행되는가?
4. `Thread`의 `run()` 메소드의 역할은?
5. `Thread`의 `start()`, `stop()` 메소드의 역할은?
6. 어떤 일이 발생하면 스레드가 실행 중지 상태로 가는가?

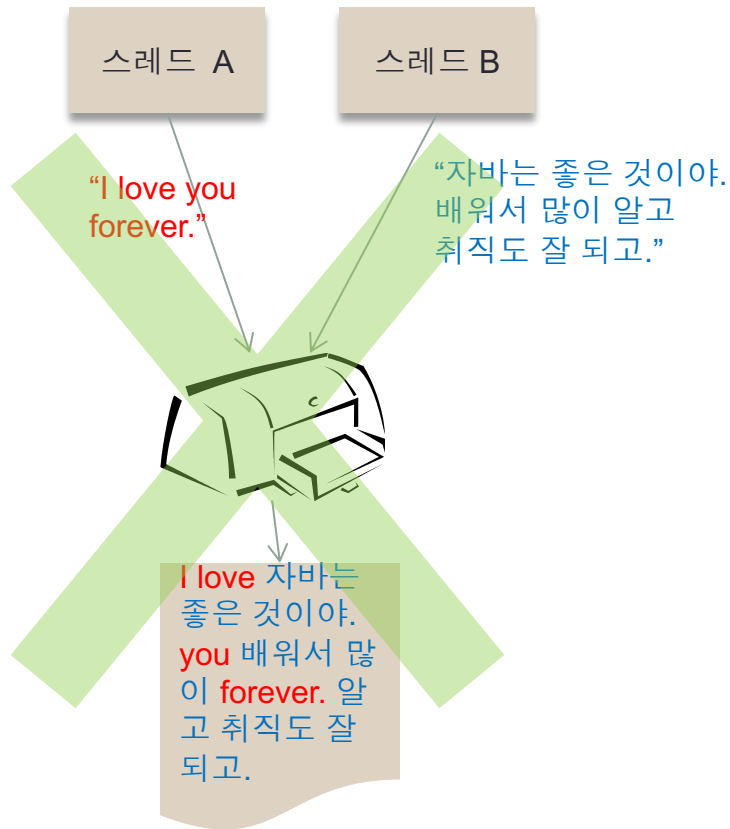
JAVA:

THREAD: SYNCHRONIZATION

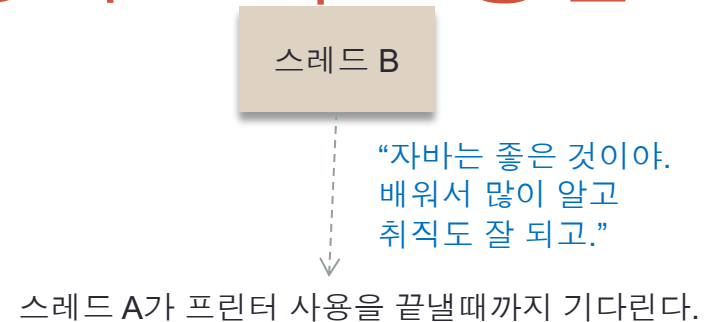
스레드 동기화(Thread Synchronization)

- 멀티스레드 프로그램 작성시 주의점
 - 다수의 스레드가 공유 데이터에 동시에 접근하는 경우
 - 공유 데이터의 값에 예상치 못한 결과 발생 가능
- 스레드 동기화
 - 멀티스레드의 공유 데이터의 동시 접근 문제 해결책
 - 공유 데이터를 접근하는 모든 스레드의 한 줄 세우기
 - 한 스레드가 공유 데이터에 대한 작업을 끝낼 때까지 다른 스레드가 대기하도록 함

두 스레드의 프린터 동시 쓰기로 충돌하는 사례

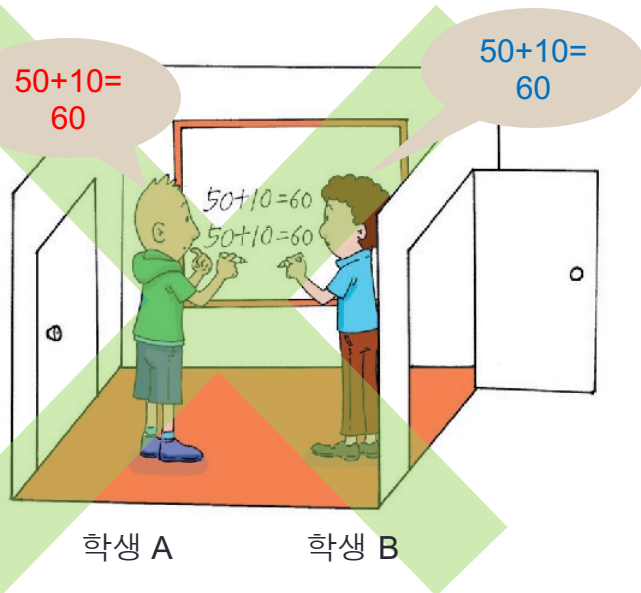


두 스레드가 동시에 프린터에 쓰는 경우
문제 발생

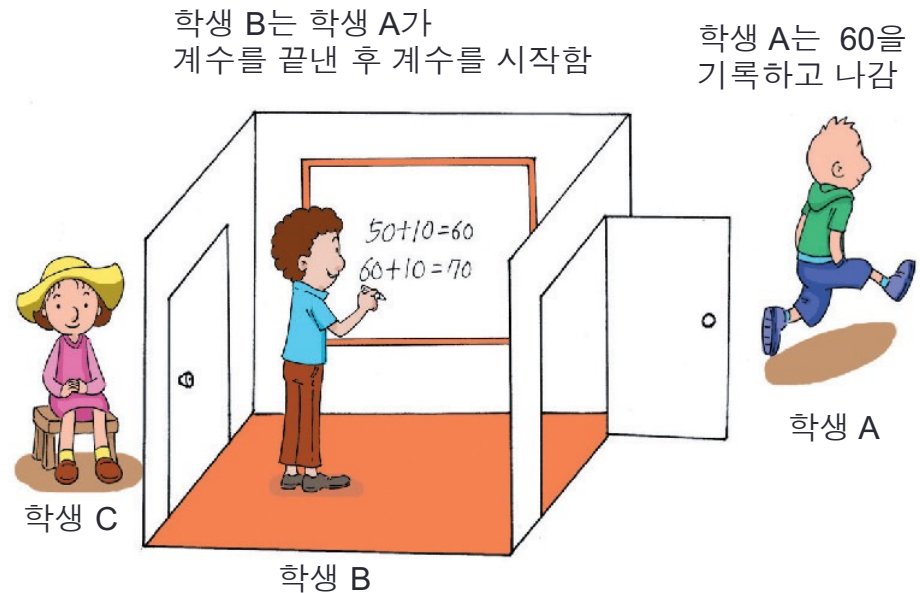


한 스레드의 출력이 끝날 때까지
대기함으로써 정상 출력

공유 집계판에 동시 접근하는 사례



두 학생이 동시에 방에 들어와서
집계판을 수정하는 경우
집계판의 결과가 잘못됨



방에 먼저 들어간 학생이
집계를 끝내기를 기다리면
정상 처리

스레드 동기화 기법

- 스레드 동기화
 - 공유 데이터에 동시에 접근하는 다수의 스레드가 공유 데이터를 배타적으로 접근하기 위해 상호 협력(coordination)하는 것
 - 동기화의 핵심
 - 스레드의 공유 데이터에 대한 배타적 독점 접근 보장
- 자바에서 스레드 동기화를 위한 방법
 - synchronized로 동기화 블록 지정
 - wait()-notify() 메소드로 스레드 실행 순서 제어

synchronized 블록 지정

- synchronized 키워드
 - 한 스레드가 독점 실행해야 하는 부분(동기화 코드)을 표시하는 키워드
 - 임계 영역(critical section) 표기 키워드
 - 메소드 전체 혹은 코드 블록
- synchronized 블록에 대한 컴파일러의 처리
 - 먼저 실행한 스레드가 모니터 소유
 - 모니터란 해당 객체를 독점적으로 사용할 수 있는 권한
 - 모니터를 소유한 스레드가 모니터를 내놓을 때까지 다른 스레드 대기

```
synchronized void add() {
    int n = getCurrentSum();
    n+=10;
    setCurrentSum(n);
}
```

synchronized 메소드

```
void execute() {
    // 다른 코드들
    //
    synchronized(this) {
        int n = getCurrentSum();
        n+=10;
        setCurrentSum(n);
    }
    //
    // 다른 코드들
}
```

synchronized 코드 블록

synchronized 사용 예 : 공유 집계판 사례 를 코딩

```

public class SynchronizedEx {
    public static void main(String [] args) {
        SharedBoard board = new SharedBoard();
        Thread th1 = new StudentThread("kitae", board);
        Thread th2 = new StudentThread("hyosoo", board);
        th1.start();
        th2.start();
    }
}

class SharedBoard {
    private int sum = 0; // 집계판의 합
    synchronized public void add() {
        int n = sum;
        Thread.yield(); // 현재 실행 중인 스레드 양보
        n += 10; // 10 증가
        sum = n; // 증가한 값을 집계함에 기록
        System.out.println(Thread.currentThread().getName() + " : " + sum);
    }
    public int getSum() { return sum; }
}

class StudentThread extends Thread {
    private SharedBoard board; // 집계판의 주소
    public StudentThread(String name, SharedBoard board) {
        super(name);
        this.board = board;
    }

    @Override
    public void run() {
        for(int i=0; i<10; i++)
            board.add();
    }
}

```

- 집계판 : class SharedBoard
- 각 학생 : class StudentThread
(각 학생은 하나의 스레드)

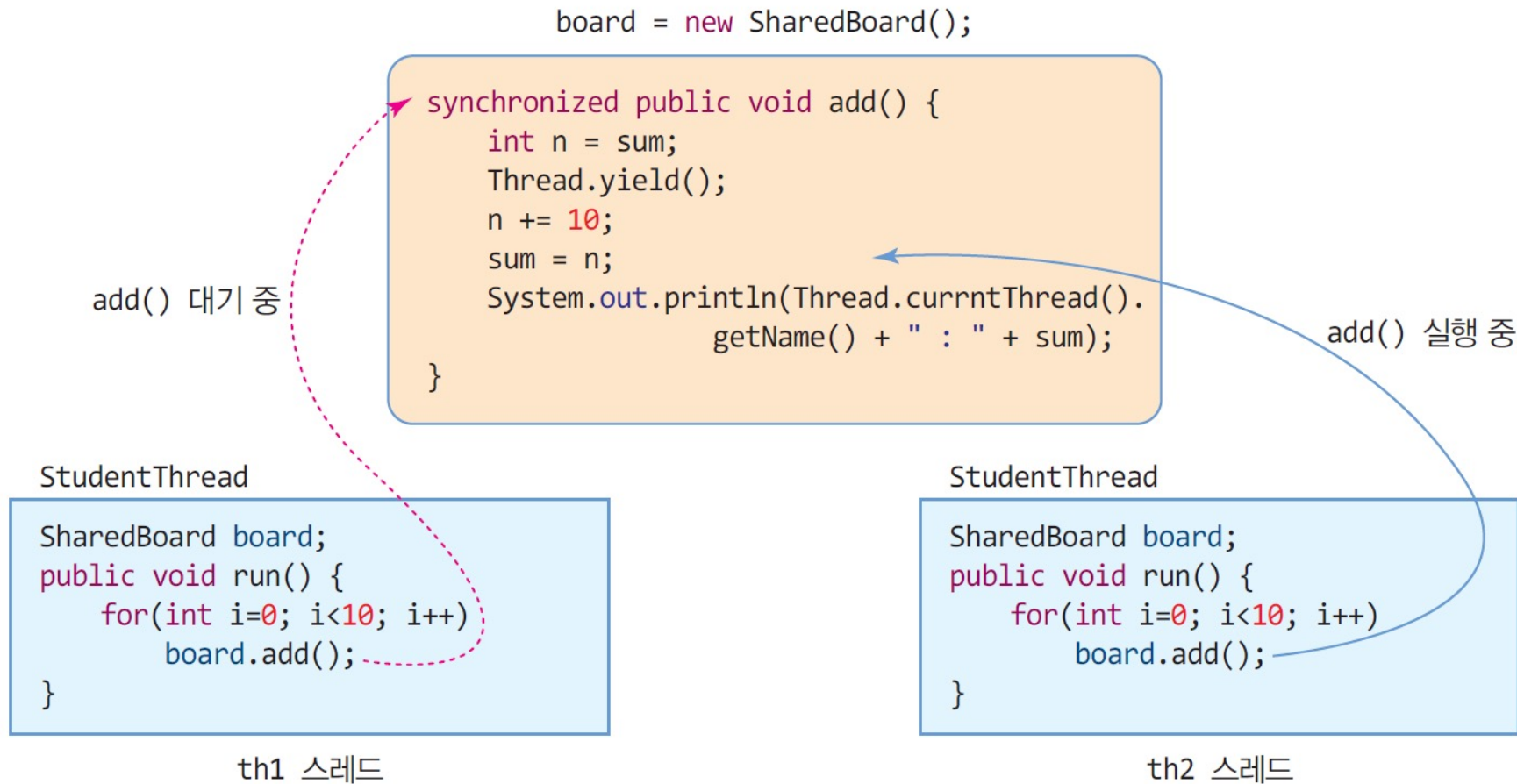
```

kitae : 10
hyosoo : 20
kitae : 30
hyosoo : 40
kitae : 50
hyosoo : 60
kitae : 70
hyosoo : 80
hyosoo : 90
hyosoo : 100
hyosoo : 110
hyosoo : 120
hyosoo : 130
hyosoo : 140
kitae : 150
kitae : 160
kitae : 170
kitae : 180
kitae : 190
kitae : 200

```

kitae와 hyosoo가 각각
10번씩 add()를 호출,
동기화가 잘 이루어져서
최종 누적 점수 sum이 200이 됨

SharedBoard의 add()를 스레드1이 실행하고 있는 동안, 스레드2가 호출하면 스레드2는 대기



공유집계판 사례에서 synchronized 사용하지 않아 충돌로 인해 데이터에 오류가 발생한 경우

```

public class SynchronizedEx {
    public static void main(String [] args) {
        SharedBoard board = new SharedBoard();
        Thread th1 = new StudentThread("kitae", board);
        Thread th2 = new StudentThread("hyosoo", board);
        th1.start();
        th2.start();
    }
}

class SharedBoard {
    private int sum = 0; // 집계판의 합
    synchronized public void add() {
        int n = sum;
        Thread.yield(); // 현재 실행 중인 스레드 양보
        n += 10; // 10 증가
        sum = n; // 증가한 값을 집계함에 기록
        System.out.println(Thread.currentThread().getName() + " : " + sum);
    }
    public int getSum() { return sum; }
}

class StudentThread extends Thread {
    private SharedBoard board; // 집계판의 주소
    public StudentThread(String name, SharedBoard board) {
        super(name);
        this.board = board;
    }

    @Override
    public void run() {
        for(int i=0; i<10; i++)
            board.add();
    }
}

```

kitae : 10
hyosoo : 20
kitae : 30 } add() 충돌
hyosoo : 30
kitae : 40
hyosoo : 50 } add() 충돌
kitae : 50 } add() 충돌
hyosoo : 60 } add() 충돌
kitae : 70 } add() 충돌
hyosoo : 70
kitae : 80
hyosoo : 90
kitae : 100 } add() 충돌
hyosoo : 100
kitae : 110
hyosoo : 120
kitae : 130
hyosoo : 140
hyosoo : 150

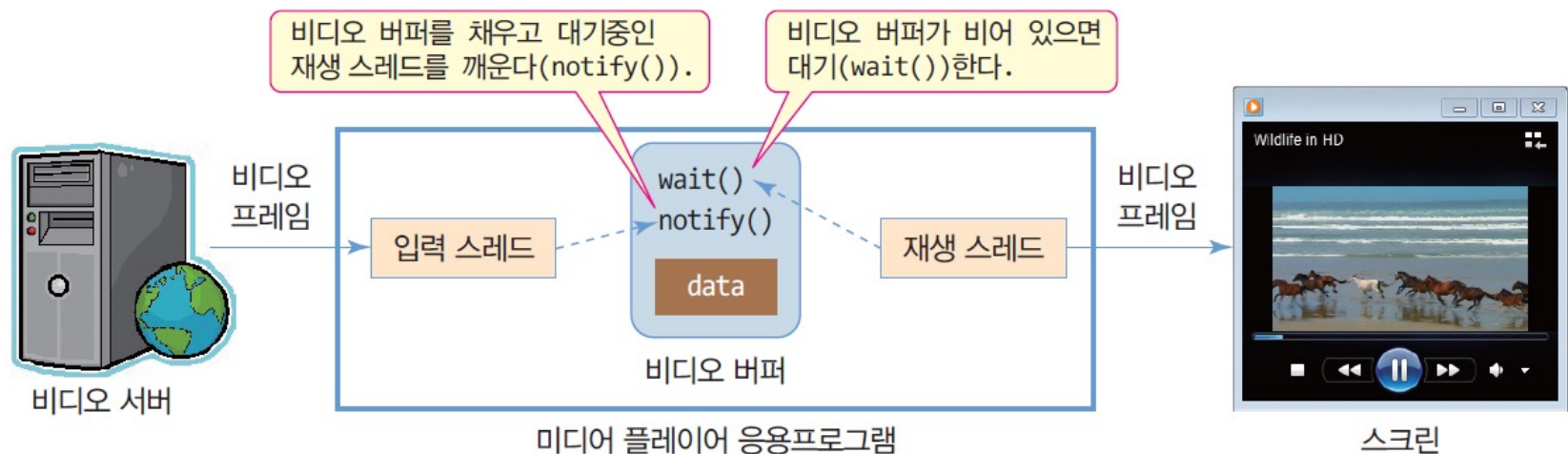
kitae와 hyosoo가 각각 10번씩 add()를 호출하였지만 add()에 대한 동기화가 이루어지지 않아 공유 변수 sum을 kitae와 hyosoo가 각각 사용하여 누적 점수가 150 밖에 되지 못함

JAVA:

THREAD: WAIT / NOTIFY

producer-consumer 문제와 동기화

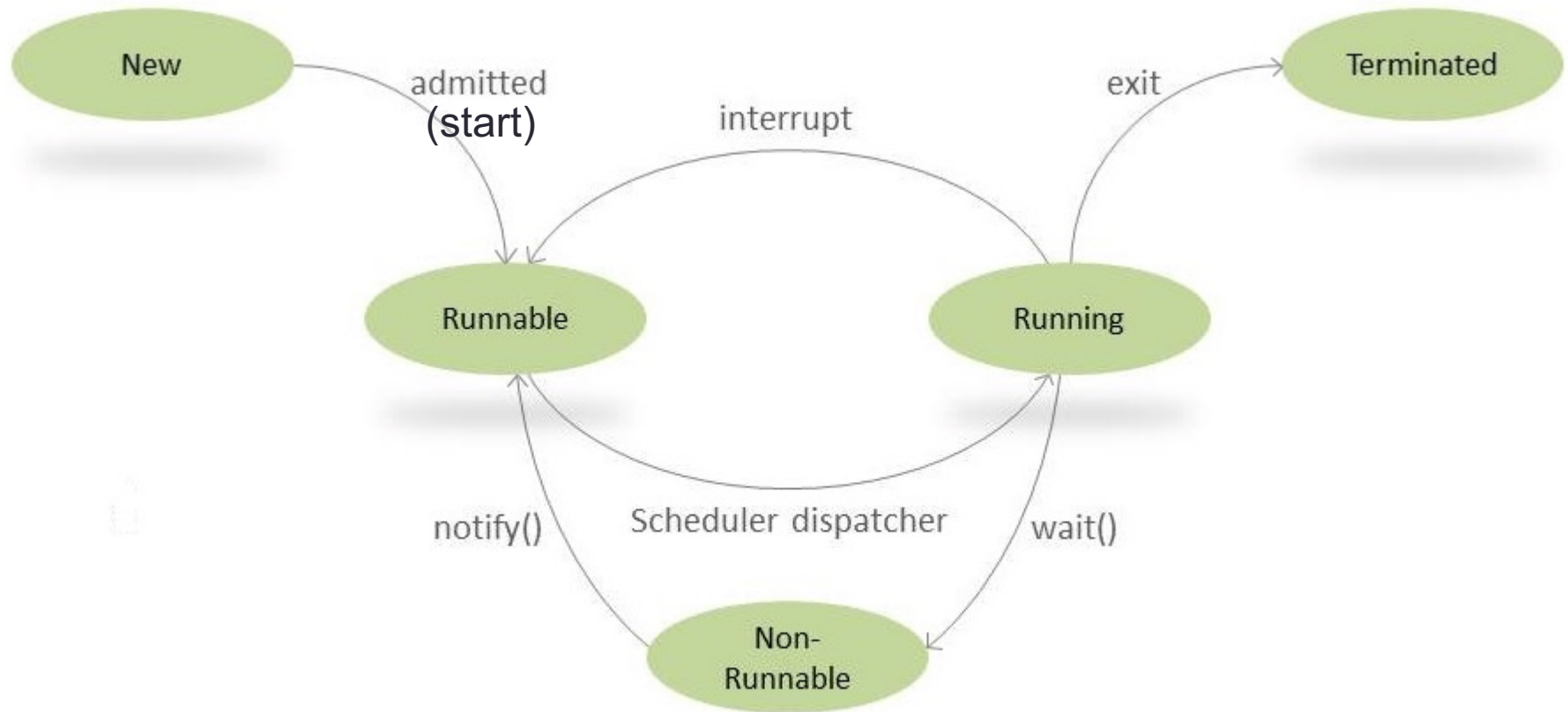
- producer-consumer 문제
 - producer : 공유 메모리에 데이터를 공급하는 스레드
 - consumer : 공유 메모리의 데이터를 소비하는 스레드
 - 문제의 본질
 - producer와 consumer 가 동시에 공유 데이터를 접근하는 문제
- producer-consumer 문제 사례
 - 미디어 플레이어
 - producer:입력스레드, consumer:재생스레드, 공유데이터:비디오버퍼



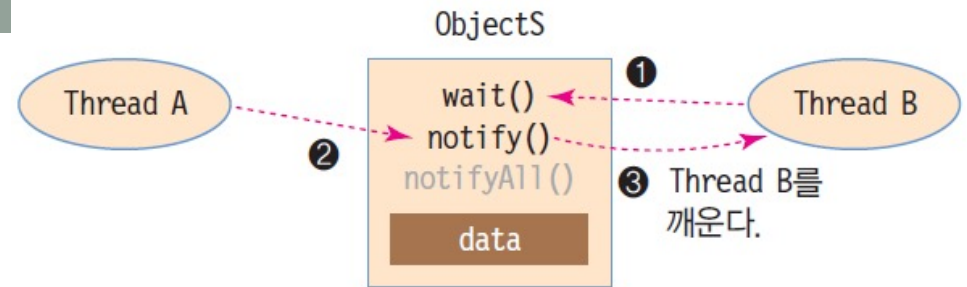
wait(), notify(), notifyAll()를 이용한 동기화

- 동기화 객체
 - 두 개 이상의 스레드 동기화에 사용되는 객체
- 동기화 메소드
 - wait()
 - 다른 스레드가 notify()를 불러줄 때까지 기다린다.
 - notify()
 - wait()를 호출하여 대기중인 스레드를 깨우고 RUNNABLE 상태로 만든다.
 - 2개 이상의 스레드가 대기중이라도 오직 한 스레드만 깨운다.
 - notifyAll()
 - wait()를 호출하여 대기중인 모든 스레드를 깨우고 모두 RUNNABLE 상태로 만든다.
 - synchronized 블록 내에서만 사용되어야 함
- wait(), notify(), notifyAll()은 Object의 메소드
 - 모든 객체가 동기화 객체가 될 수 있다.
 - Thread 객체도 동기화 객체로 사용될 수 있다.

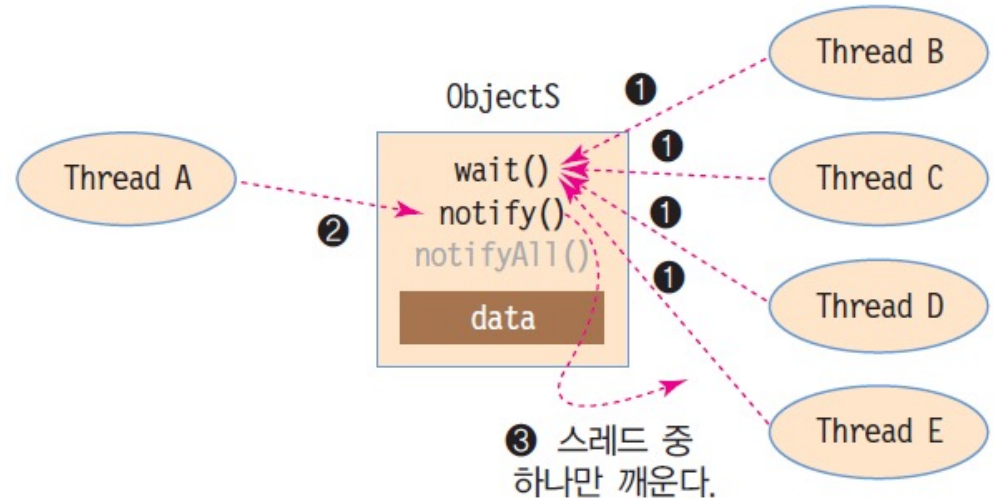
Lifecycle of a Thread



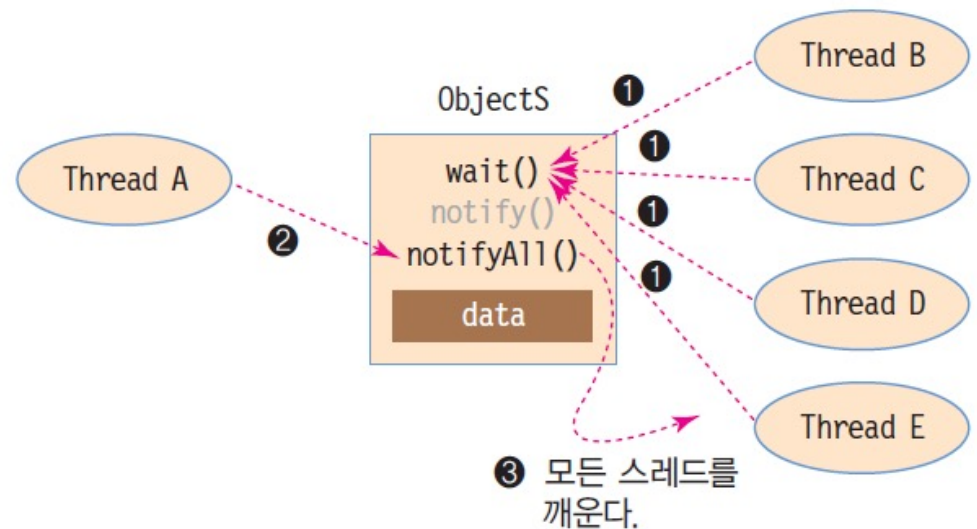
Thread A가 ObjectS.wait()를 호출하여 무한 대기하고, Thread B가 ObjectS.notify()를 호출하여 ObjectS에 대기하고 있는 Thread A를 깨운다.



4 개의 스레드가 모두 ObjectS.wait()를 호출하여 대기하고, ThreadA는 ObjectS.notify()를 호출하여 대기 중인 스레드 중 하나만 깨우는 경우



4 개의 스레드가 모두 ObjectS.wait()를 호출하여 대기하고, ThreadA는 ObjectS.notifyAll()를 호출하여 대기 중인 4개의 스레드를 모두 깨우는 경우



예제 13-6 : wait(), notify()를 이용한 바 채우기

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class MyLabel extends JLabel {
    int barSize = 0; // 바의 크기
    int maxBarSize;

    MyLabel(int maxBarSize) {
        this.maxBarSize = maxBarSize;
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.MAGENTA);
        int width = (int)((double)(this.getWidth()))
            / maxBarSize * barSize;
        if (width == 0) return;
        g.fillRect(0, 0, width, this.getHeight());
    }

    synchronized void fill() {
        if (barSize == maxBarSize) {
            try {
                wait();
            } catch (InterruptedException e) { return; }
        }
        barSize++;
        repaint(); // 바 다시 그리기
        notify();
    }
}
```

```
synchronized void consume() {
    if (barSize == 0) {
        try {
            wait();
        } catch (InterruptedException e) {
            return;
        }
        barSize--;
        repaint(); // 바 다시 그리기
        notify();
    }
}

class ConsumerThread extends Thread {
    MyLabel bar;

    ConsumerThread(MyLabel bar) {
        this.bar = bar;
    }

    public void run() {
        while (true) {
            try {
                sleep(200);
                bar.consume();
            } catch (InterruptedException e) {
                return;
            }
        }
    }
}
```

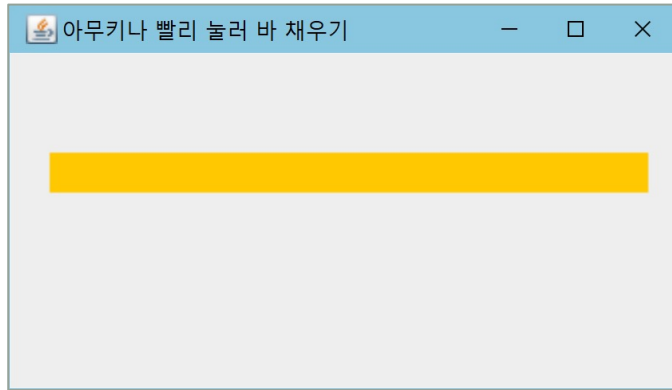
```
public class TabAndThreadEx extends
JFrame {
    MyLabel bar = new MyLabel(100);
    TabAndThreadEx(String title) {
        super(title);
        this.setDefaultCloseOperation
            (JFrame.EXIT_ON_CLOSE);
        Container c = getContentPane();
        c.setLayout(null);
        bar.setBackground(Color.ORANGE);
        bar.setOpaque(true);
        bar.setLocation(20, 50);
        bar.setSize(300, 20);
        c.add(bar);

        c.addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent e)
            {
                bar.fill();
            }
        });
        setSize(350, 200);
        setVisible(true);

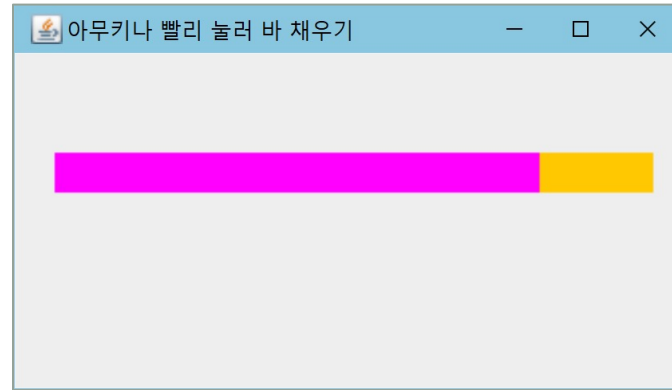
        c.requestFocus();
        ConsumerThread th = new
            ConsumerThread(bar);
        th.start(); // 스레드 시작
    }

    public static void main(String[] args) {
        new TabAndThreadEx(
            "아무키나 빨리 눌러 바 채우기");
    }
}
```

실행 결과



초기 화면



키를 반복하여 빨리 누른 화면