# 쓰레드와 멀티태스킹

comsi.java@gmail.com
박 경 태

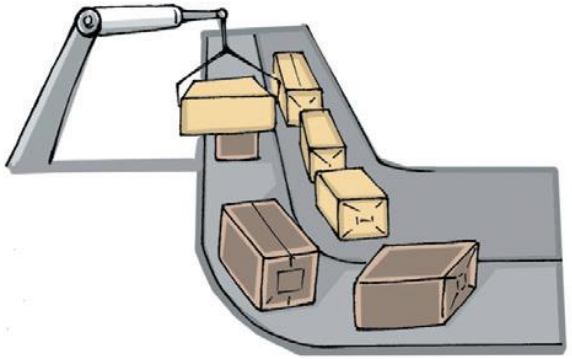
http://github.com/hopypark

# 멀티태스킹(multi-tasking) 개념

- 멀티태스킹
  - 하나의 응용프로그램이 여러 개의 작업(태스크)을 동시에 처리

다림질하면서 이어폰으로
전화하는 주부

운전하면서
화장하는 운전자



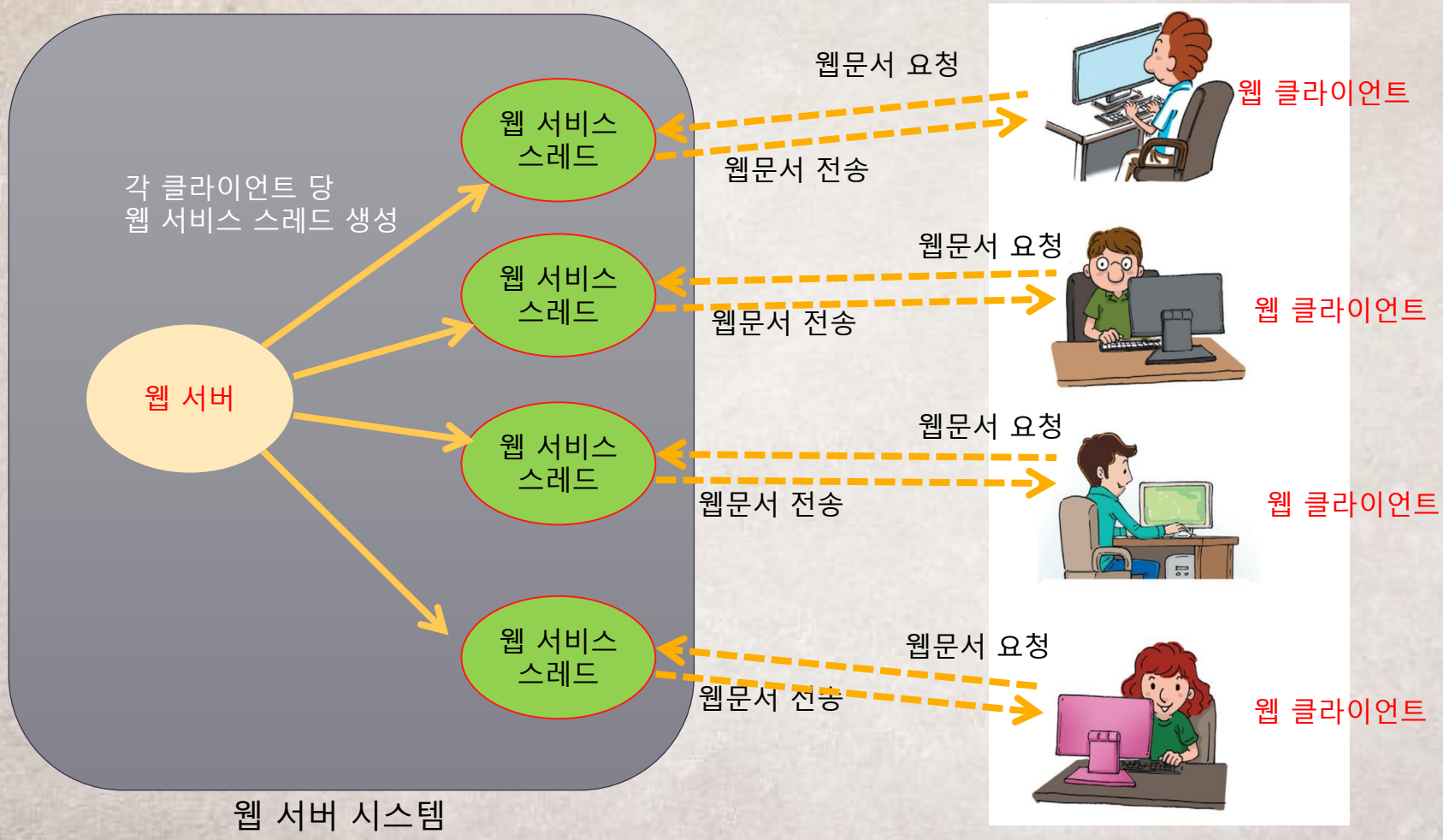
제품의 판독과 포장 작업의
두 기능을 갖춘 기계

# 쓰레드와 멀티스레딩

- 쓰레드
  - 운영체제에 의해 관리되는 하나의 작업 혹은 태스크
  - 다수의 쓰레드를 동시에 실행시키도록 응용프로그램을 작성하는 기법을 멀티쓰레딩이라 함.
  - 쓰레드 실행코드와 운영체제에서 관리하는 스레드 정보로 이분화
    - 스레드 코드 - 작업을 실행하기 위해 사용자가 작성한 프로그램 코드
    - 스레드 정보 - 스레드 명, 스레드 ID, 스레드 소요시간, 스레드 우선 순위 등 운영체제가 관리하는 정보

- 자바의 멀티태스킹
  - 멀티스레딩만 가능
    - 자바에 프로세스 개념은 존재하지 않고, 스레드 개념만 존재
    - 스레드는 실행 단위
    - 스레드는 스케쥴링 단위
  - 하나의 응용프로그램은 여러 개의 스레드로 구성 가능
    - 스레드 사이의 통신에 따른 오버헤드가 크지 않음

# 웹 서버의 멀티스레딩 사례



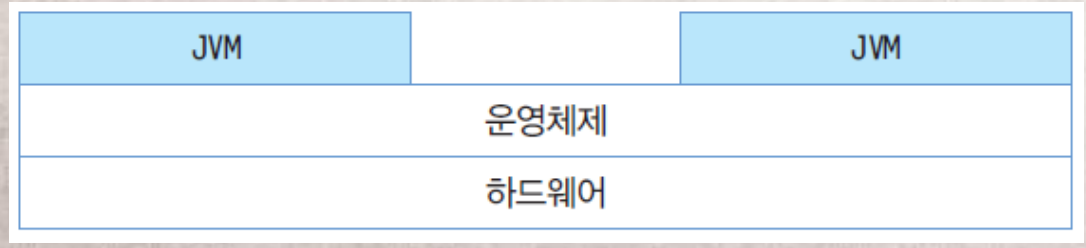
# 자바 스레드(Thread)란?

- 자바 스레드
  - 자바 가상 기계(JVM)에 의해 스케쥴되는 실행 단위의 코드 블럭
  - 스레드의 생명 주기는 JVM에 의해 관리됨
    - JVM은 스레드 단위로 스케쥴링

- JVM과 멀티스레드의 관계
  - 하나의 JVM은 하나의 자바 응용프로그램만 실행
    - 자바 응용프로그램이 시작될 때 JVM이 함께 실행됨
    - 자바 응용프로그램이 종료하면 JVM도 함께 종료함
  - 하나의 응용프로그램은 하나 이상의 스레드로 구성 가능

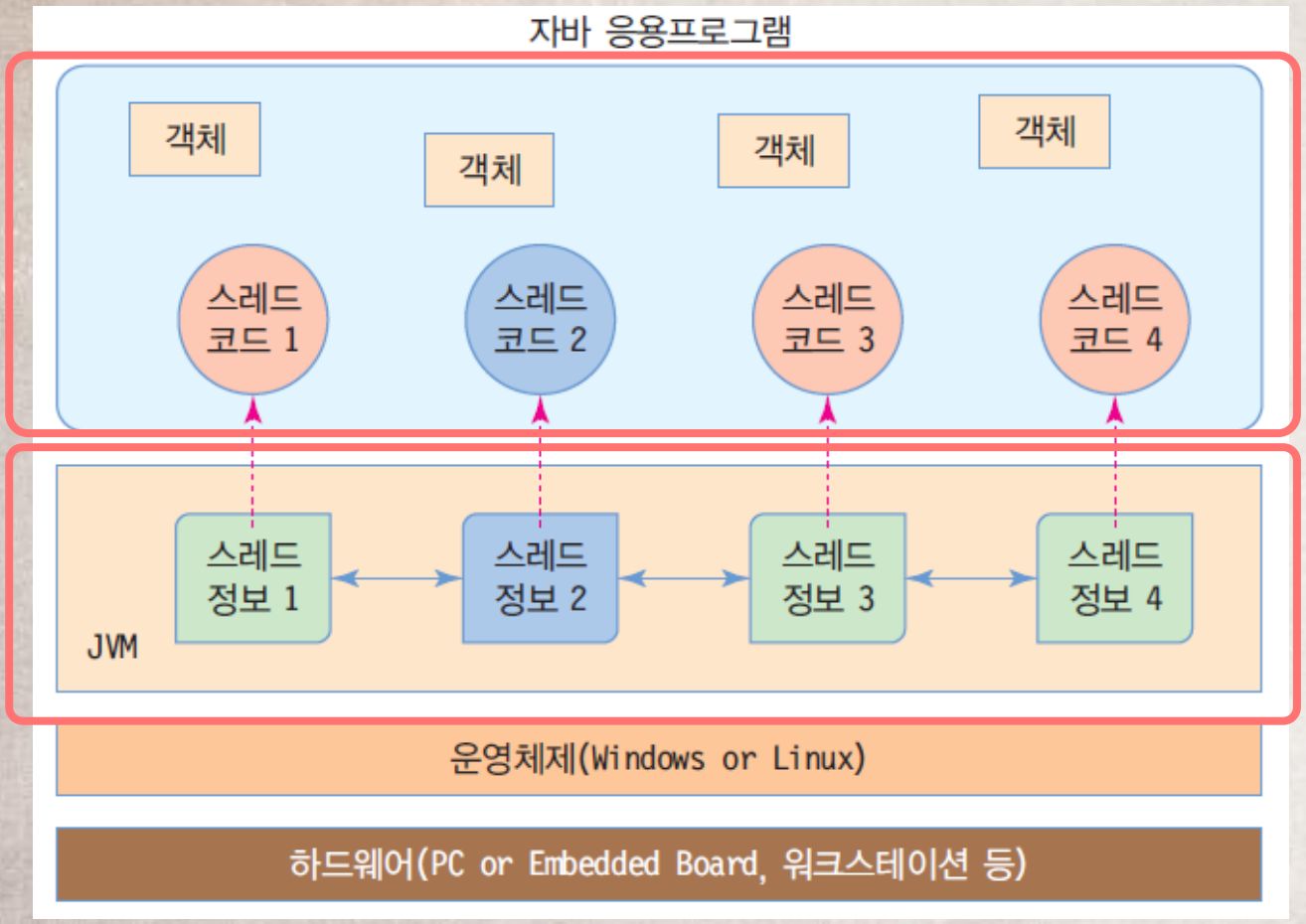
# JVM과 자바 응용프로그램, 스레드의 관계


(1) JVM은 한 개의 응용 프로그램만 실행 가능


두 개의 자바 응용프로그램을 동시에 실행시키고자 하면 두 개의 JVM을 이용하고 응용프로그램은 서로 소켓 등을 이용하여 통신

# 자바 스레드와 JVM




- 각 스레드의 스레드 코드는 응용프로그램 내에 존재함

JVM이 스레드를 관리함
- 스레드가 몇 개인지?
- 스레드 코드의 위치가 어디인지?
- 스레드의 우선순위는 얼마인지?

등 …

현재 하나의 JVM에 의해 4 개의 스레드가 실행 중이며
그 중 스레드 2가 JVM에 의해 스케쥴링되어 실행되고 있음

# 스레드 만들기

- 스레드 실행을 위해 개발자가 하는 작업
  - 스레드 코드 작성
  - JVM에게 스레드를 생성하고 스레드 코드를 실행하도록 요청

- 자바에서 스레드 만드는 2 가지 방법
  - java.lang.Thread 클래스를 이용하는 경우
  - java.lang.Runnable 인터페이스를 이용하는 경우

# Thread 클래스의 메소드

- 생성자
  - Thread()
  - Thread(Runnable target)
  - Thread(String name)
  - Thread(Runnable target, String name)
- 스레드 시작시키기
  - void start()
- 스레드 코드
  - void run()
- 스레드 잠자기
  - static void sleep(long mills)
- 다른 스레드 죽이기
  - void interrupt()
- 다른 스레드에게 양보
  - static void yield()
  - 현재 스레드의 실행을 중단하고 다른 스레드가 실행될 수 있도록 양보한다.
- 다른 스레드가 죽을 때까지 기다리기
  - void join()
- 현재 스레드 객체 알아내기
  - static Thread currentThread()
- 스레드 ID 알아내기
  - long getId()
- 스레드 이름 알아내기
  - String getName()
- 스레드 우선순위값 알아내기
  - int getPriority()
- 스레드의 상태 알아내기
  - Thread.State getState()
    - Blocked, Waiting, Runnable, ..

# Thread 클래스를 이용한 스레드 생성

- 스레드 클래스 작성
  - Thread 클래스 상속. 새 클래스 작성
- 스레드 코드 작성
  - run() 메소드 오버라이딩
    - run() 메소드를 스레드 코드라고 부름
    - run() 메소드에서 스레드 실행 시작
- 스레드 객체 생성
- 스레드 시작
  - start() 메소드 호출
    - 스레드로 작동 시작
    - JVM에 의해 스케쥴되기 시작함

```
class TimerThread extends Thread {
   ..................................................


}
```

```
TimerThread th = new TimerThread();
```

```
th.start();
```

# * Thread를 상속받아 1초 단위로 초 시간을 출력하는 TimerThread 스레드 작성

스레드 클래스 정의

스레드 코드 작성

1초에 한 번씩 n을 증가 시켜 콘솔에 출력한다.

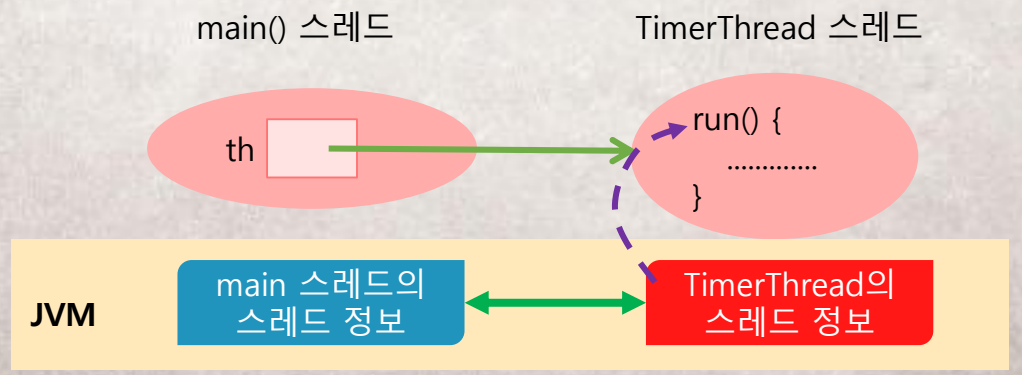
```
class TimerThread extends Thread {
    int n = 0;
    public void run() {
        while(true) { // 무한루프를 실행한다
            System.out.println(n);
            n++;
            try {
                sleep(1000); //1초 동안 잠을 잔 후 깨어난다.
            }
            catch(InterruptedException e){return;}
        }
    }
}
```

스레드 객체 생성

스레드 시작

```
public class TestThread {
    public static void main(String [] args) {
        TimerThread th = new TimerThread();
        th.start();
    }
}
```

```
0
1
2
3
4
```



# 예제 09-1 : Thread를 상속받아 1초 단위의 타이머 만들기

TimerThread.java | TimerThreadEx.java

```
1
2  public class TimerThread extends Thread {
3
4      @Override
5      public void run() {
6          int n = 0;
7
8          while(true) {
9              System.out.println(n);
10             n++;
11             try {
12                 Thread.sleep(1000);
13             }catch(InterruptedException e) {
14                 e.printStackTrace();
15                 return;
16             }
17         }
18     }
19
20 }
21
```

TimerThread.java | *TimerThreadEx.java

```
1
2  public class TimerThreadEx {
3
4      public TimerThreadEx() {
5          TimerThread th = new TimerThread();
6          th.start();
7      }
8
9      public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         new TimerThreadEx();
12     }
13
14 }
15
16
```

Problems | Javadoc | Declaration | Console

```
<terminated> RunnableTimerEx [Java Application] C:₩Program Files₩Java₩jre7₩bin₩javaw.exe (2015. 5. 14. 오후 8:27:32)
0
1
2
3
4
5
```

# 스레드 주의 사항

- run() 메소드가 종료하면 스레드는 종료한다.
  - 스레드가 계속 존재하게 하려면 run() 메소드 내에 무한 루프가 실행되어야 한다.
- 한번 종료한 스레드는 다시 시작시킬 수 없다.
  - 스레드 객체를 생성하여 다시 스레드로 등록하여야 한다.
- 한 스레드에서 다른 스레드를 강제 종료할 수 있다.
  - 뒤에서 다룸

# Runnable 인터페이스로 스레드 만들기

- 스레드 클래스 작성
  - Runnable 인터페이스 구현하는 새 클래스 작성
- 스레드 코드 작성
  - run() 메소드 오버라이딩
    - run() 메소드를 스레드 코드라고 부름
    - run() 메소드에서 스레드 실행 시작
- 스레드 객체 생성
- 스레드 시작
  - start() 메소드 호출

```
class TimerRunnable implements Runnable {



}
```

```
Thread th =  new Thread(new TimerRunnable());
```

```
th.start();
```

# *Runnable 인터페이스를 상속받아 1초 단위로 초 시간을 출력하는 스레드 작성

스레드 클래스 정의

스레드 코드 작성

1초에 한 번씩 n을 증가 시켜 콘솔에 출력한다.

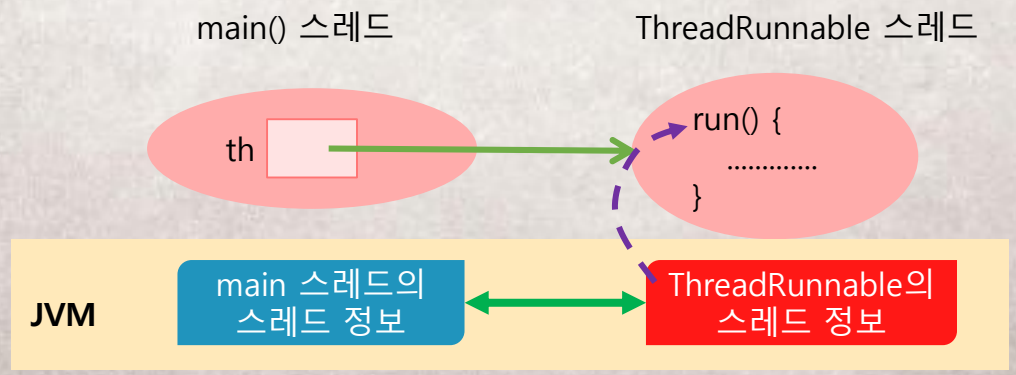
```
class TimerRunnable implements Runnable {
   int n = 0;
   public void run() {
      while(true) { // 무한루프를 실행한다
         System.out.println(n);
         n++;
          try {
             sleep(1000); //1초 동안 잠을 잔 후 깨어난다.
         }
         catch(InterruptedException e){return;}
      }
   }
}
```

스레드 객체 생성

스레드 시작

```
public class TestRunnable{
   public static void main(String [] args) {
      Thread th = new Thread(new TimerRunnable());
      th.start();
   }
}
```

```
0
1
2
3
4
```



# 예제 09-2 : Runnable인터페이스를 구현하여 1초 단위 타이머 만들기

RunnableTimer.java

```
1
2  public class RunnableTimer implements Runnable {
3
4      @Override
5      public void run() {
6          int n = 0;
7
8          while(true) {
9              System.out.println(n);
10             n++;
11             try {
12                 Thread.sleep(1000);
13             }catch(InterruptedException e) {
14                 e.printStackTrace();
15                 return;
16             }
17         }
18     }
19
20 }
21
```

RunnableTimerEx.java

```
1
2  public class RunnableTimerEx {
3
4      public RunnableTimerEx() {
5          RunnableTimer runnable = new RunnableTimer();
6          Thread th = new Thread(runnable);
7          th.start();
8      }
9
10     public static void main(String[] args) {
11         new RunnableTimerEx();
12     }
13 }
14
```

Problems | Javadoc | Declaration | Console

```
<terminated> RunnableTimerEx [Java Application] C:₩Program Files₩Java₩jre7₩bin₩javaw.exe (2015. 5. 14. 오후 8:27:32)
0
1
2
3
4
5
```

# 스레드 정보

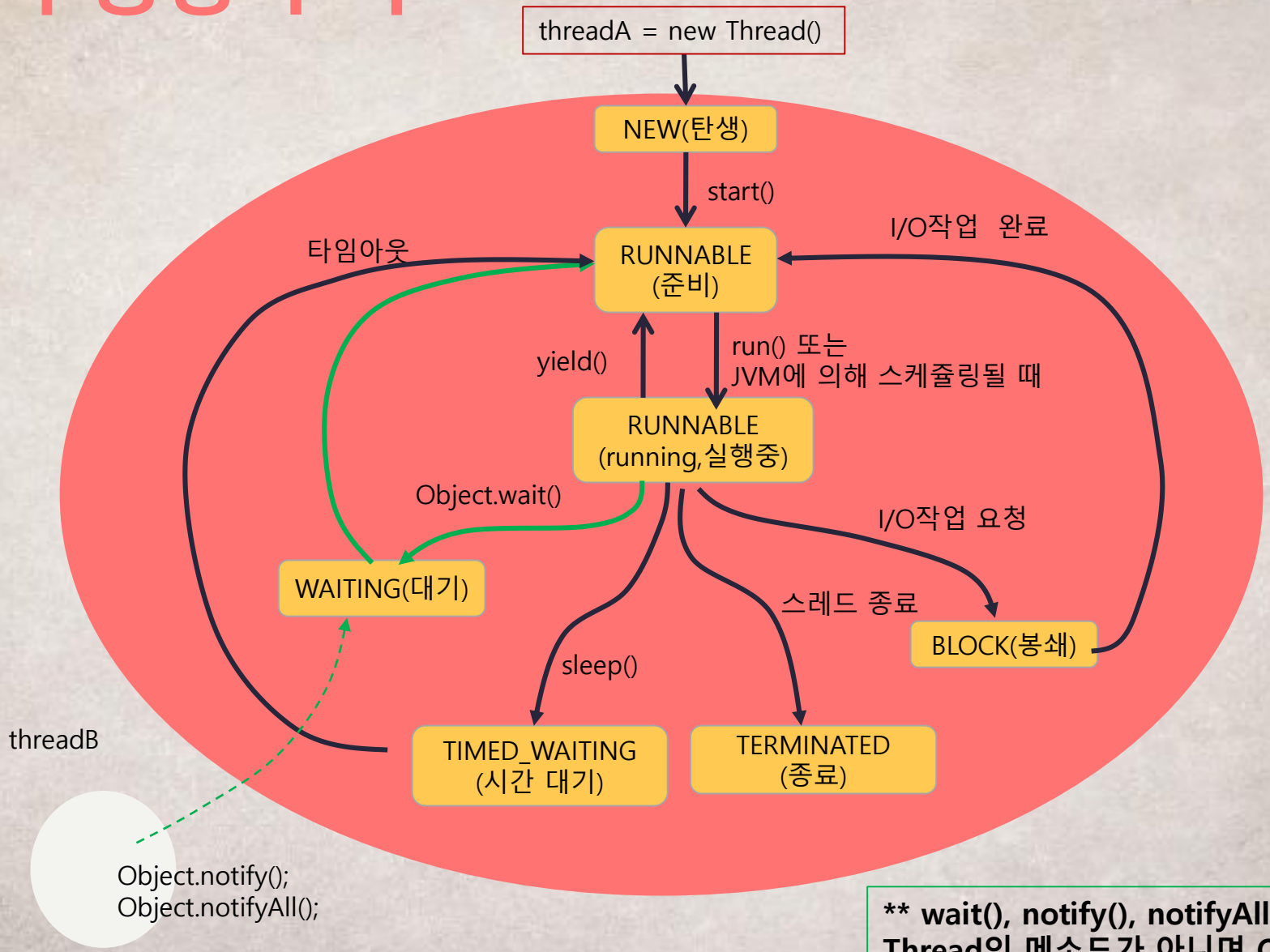
| 필 드 | 타 입 | 내 용 |
|---|---|---|
| 스레드 이름 | 스트링 | 스레드의 이름으로서 사용자가 지정 |
| 스레드 ID | 정수 | 스레드 고유의 식별자 번호 |
| 스레드 상태 | 정수 | NEW, RUNNABLE, WAITING, TIMED_WAITING, BLOCK,<br>TERMINATED 등 6개 상태 중 하나<br>- TIMED_WAITING은 시간 인자가 들어간 메소드(Thread.sleep())가 호출될 때,<br>진입되는 상태 |
| 스레드 우선순위 | 정수 | 스레드 스케줄링 시 사용되는 우선순위 값으로서<br>1~10 사이의 값이며 10이 최상위 우선순위 |
| 스레드 그룹 | 정수 | 여러 개의 자바 스레드가 하나의 그룹을 형성할 수<br>있으며 이 경우 스레드가 속한 그룹 |
| 스레드 레지스터 스택 | 메모리 블록 | 스레드가 실행되는 동안 레지스터들의 값 |

# 스레드 상태

- 스레드 상태 6 가지
  - NEW
    - 스레드가 생성되었지만 스레드가 아직 실행할 준비가 되지 않았음
  - RUNNABLE
    - 스레드가 JVM에 의해 실행되고 있거나 실행 준비되어 스케쥴링을 기다리는 상태
  - WAITING
    - 어떤 Object 객체에서 다른 스레드가 notify(), notifyAll()을 불러주기를 기다리고 있는 상태.
    - 스레드 동기화를 위해 사용
  - TIMED_WAITING
    - 스레드가 sleep(n) 호출로 인해 n 밀리초 동안 잠을 자고 있는 상태
  - BLOCK
    - 스레드가 I/O 작업을 요청하면 JVM이 자동으로 이 스레드를 BLOCK 상태로 만든다.
  - TERMINATED
    - 스레드가 종료한 상태
- 스레드 상태는 JVM에 의해 기록 관리됨

# 스레드 상태와 생명 주기

**스레드 상태 6 가지**
- NEW
- RUNNABLE
- WAITING
- TIMED_WAITING
- BLOCK
- TERMINATED




**** wait(), notify(), notifyAll()은**
**Thread의 메소드가 아니며 Object의 메소드임**

# 스레드 우선 순위와 스케쥴링

- 스레드의 우선 순위
  - 최대값 = 10(MAX_PRIORITY)
  - 최소값 = 1(MIN_PRIORITY)
  - 보통값 = 5(NORMAL_PRIORITY)
- 스레드 우선 순위는 응용프로그램에서 변경 가능
  - void setPriority(int priority)
  - int getPriority()
- main() 스레드의 우선 순위 값은 초기에 보통값 5로 지정.
- 스레드는 부모 스레드와 동일한 우선순위 값을 가지고 탄생
- JVM의 스케쥴링 정책
  - 철저한 우선 순위 기반
    - 가장 높은 우선 순위의 스레드가 우선적으로 스케쥴링
    - 동일한 우선 순위의 스레드는 돌아가면서 스케쥴링(라운드 로빈).

# main()은 자바의 main 스레드

- main() 메소드
  - JVM에 의해 자동으로 스레드화
  - 자바 스레드 : main 스레드
  - main() 함수가 스레드 코드로 사용

TimerRunnable.java | RunnableTimerEx.java | ThreadMainEx.java

```
1
2  public class ThreadMainEx {
3
4      public static void main(String[] args) {
5          long id = Thread.currentThread().getId();
6          String name = Thread.currentThread().getName();
7          int priority = Thread.currentThread().getPriority();
8          Thread.State state = Thread.currentThread().getState();
9
10         System.out.println("현재 스레드 이름 = " + name);
11         System.out.println("현재 스레드 ID = " + id);
12         System.out.println("현재 스레드 우선순위 값 = " + priority);
13         System.out.println("현재 스레드 상태 = " + state);
14     }
15
16 }
```

Problems | Javadoc | Declaration | Console

```
<terminated> ThreadMainEx [Java Application] C:₩Program Files₩Java₩jre7₩bin₩javaw.exe (2015. 5. 14. 오후 8:32:40)
현재 스레드 이름 = main
현재 스레드 ID = 1
현재 스레드 우선순위 값 = 5
현재 스레드 상태 = RUNNABLE
```

# 스레드 종료와 타 스레드 강제 종료

- 스스로 종료
  - run() 메소드 리턴
- 타 스레드에서 강제 종료 : interrupt() 메소드 사용

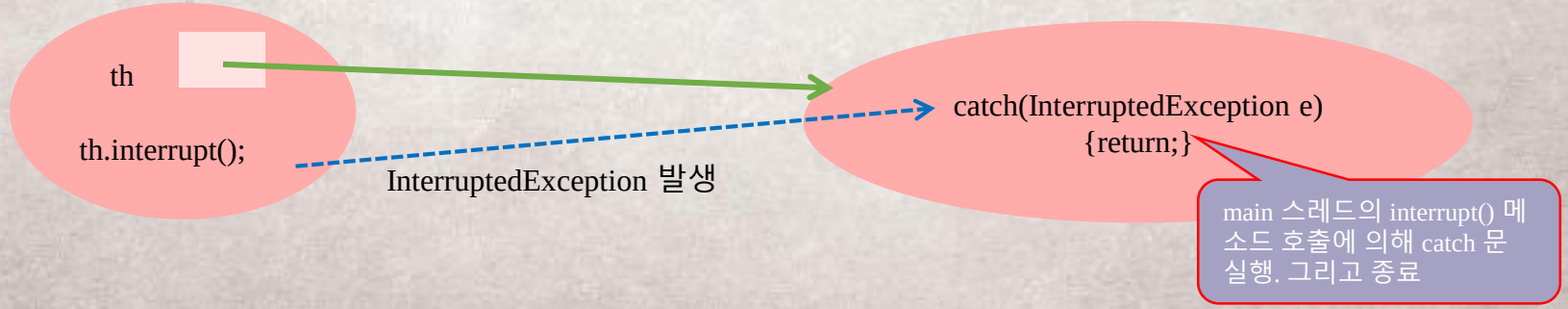
```
public static void main(String [] args) {
    TimerThread th = new  TimerThread();
    th.start();

    th.interrupt(); // TimerThread 강제 종료
}
```

main() 스레드

```
class TimerThread extends Thread {
    int n = 0;
    public void run() {
        while(true) {
            System.out.println(n); // 화면에 카운트 값 출력
            n++;
            try {
                sleep(1000);
            }
            catch(InterruptedException e){
                return; // 예외를 받고 스스로 리턴하여 종료
            }
        }
    }
}
```

만일 return 하지 않으면 스레드는 종료하지 않음

TimerThread 스레드

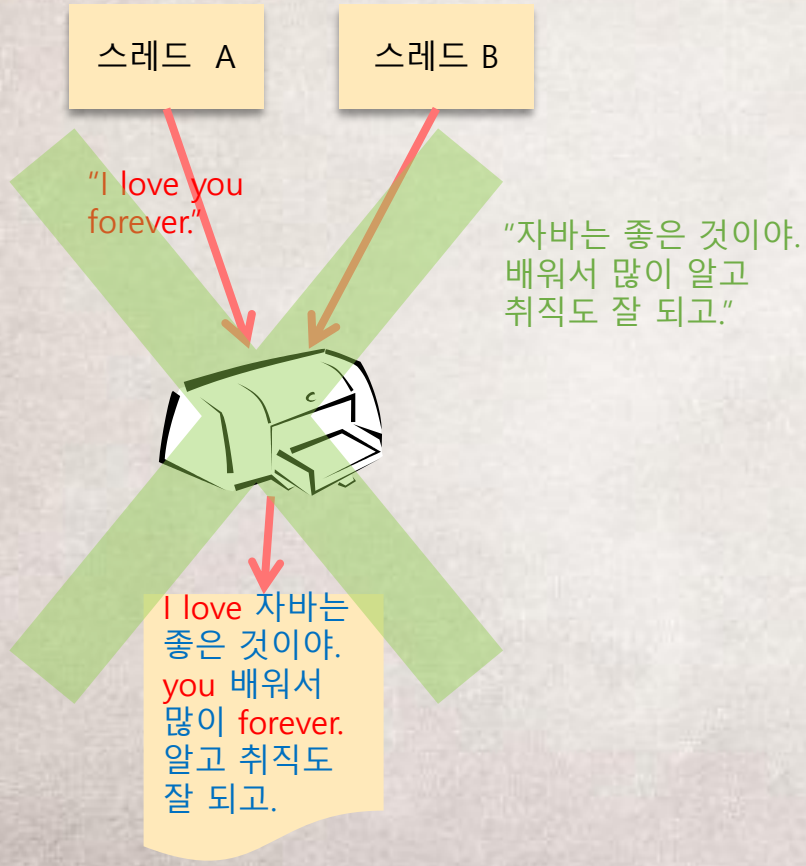


# 스레드 동기화(Thread Synchronization)

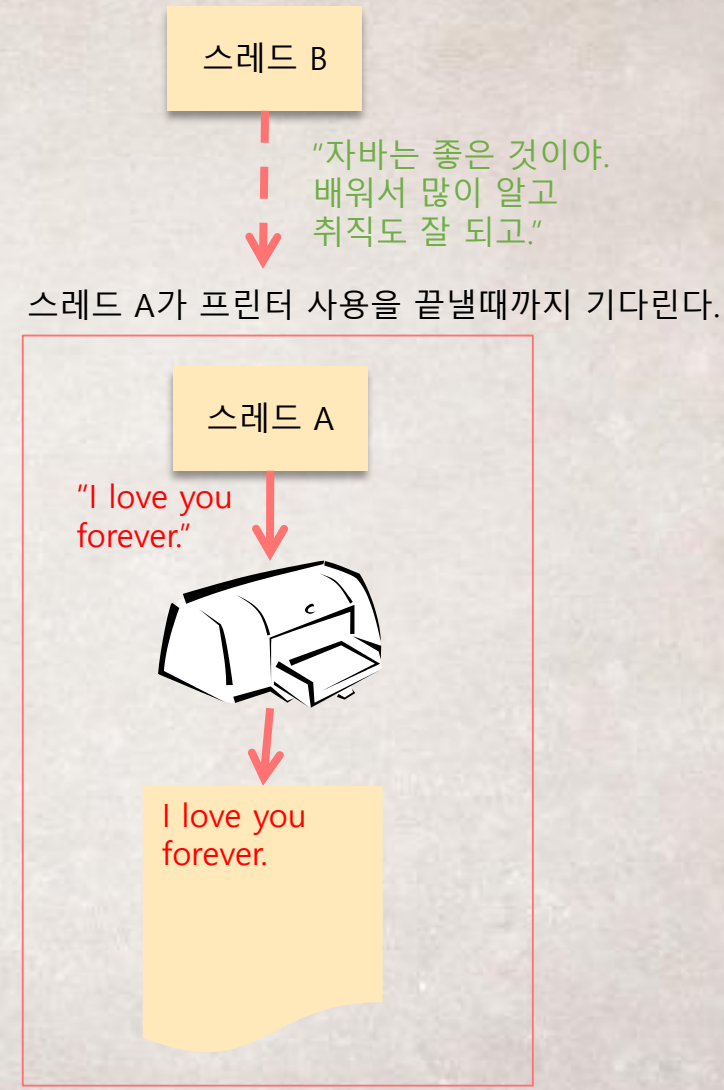
- 멀티스레드 프로그램 작성시 주의점
  - 다수의 스레드가 공유 데이터에 동시에 접근하는 경우
    - 공유 데이터의 값에 예상치 못한 결과 발생 가능

- 스레드 동기화
  - 멀티스레드의 공유 데이터의 동시 접근 문제 해결책
    - 공유데이타를 접근하고자 하는 모든 스레드의 한 줄 세우기
    - 한 스레드가 공유 데이터에 대한 작업을 끝낼 때까지 다른 스레드가 공유 데이터에 접근하지 못하도록 함

# 두 스레드가 프린터에 동시 쓰기 수행 시




두 개의 스레드가 동시에 프린터에 쓰는 경우 문제 발생




두 개의 스레드가 순서를 지켜 프린터에 쓰는 경우 정상 출력

# synchronized 키워드

- synchronized 키워드
  - 한 스레드만이 독점적으로 실행되어야 하는 부분(동기화 코드)을 표시하는 키워드
    - 임계 영역(ciritical section) 표기 키워드
- **synchronized 키워드 사용 가능한 부분**
  - **메소드 전체 혹은 코드 블럭**
- synchronized 부분이 실행될 때,
  - 실행 스레드는 모니터 소유
    - 모니터란 해당 객체를 독점적으로 사용할 수 있는 권한
  - 모니터를 소유한 스레드가 모니터를 내놓을 때까지 다른 스레드는 대기

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

synchoronized 메소드

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

synchoronized 코드블럭

# 09.3. synchronized 사용 예 : 점수판 코딩

SyncObject.java | WorkerThread.java | SynchronizedEx.java

```
1
2  public class SynchronizedEx {
3
4      public static void main(String[] args) {
5          SyncObject obj = new SyncObject();
6
7          Thread th1 = new WorkerThread("kitae",obj);
8          Thread th2 = new WorkerThread("hyosoo",obj);
9          th1.start();
10         th2.start();
11     }
12
13 }
14
```

SyncObject.java | WorkerThread.java | SynchronizedEx.java

```
1
2  class SyncObject {
3      int sum = 0;
4
5      synchronized void add(){
6          int n = sum;
7
8          Thread.currentThread().yield();
9
10         n += 10;
11         sum = n;
12         System.out.println(Thread.currentThread().getName() + ":" + sum);
13     }
14
15 }
16
```

# 09.3 synchronized 사용 예 : 집계판 사례를 코딩

SyncObject.java | WorkerThread.java | SynchronizedEx.java

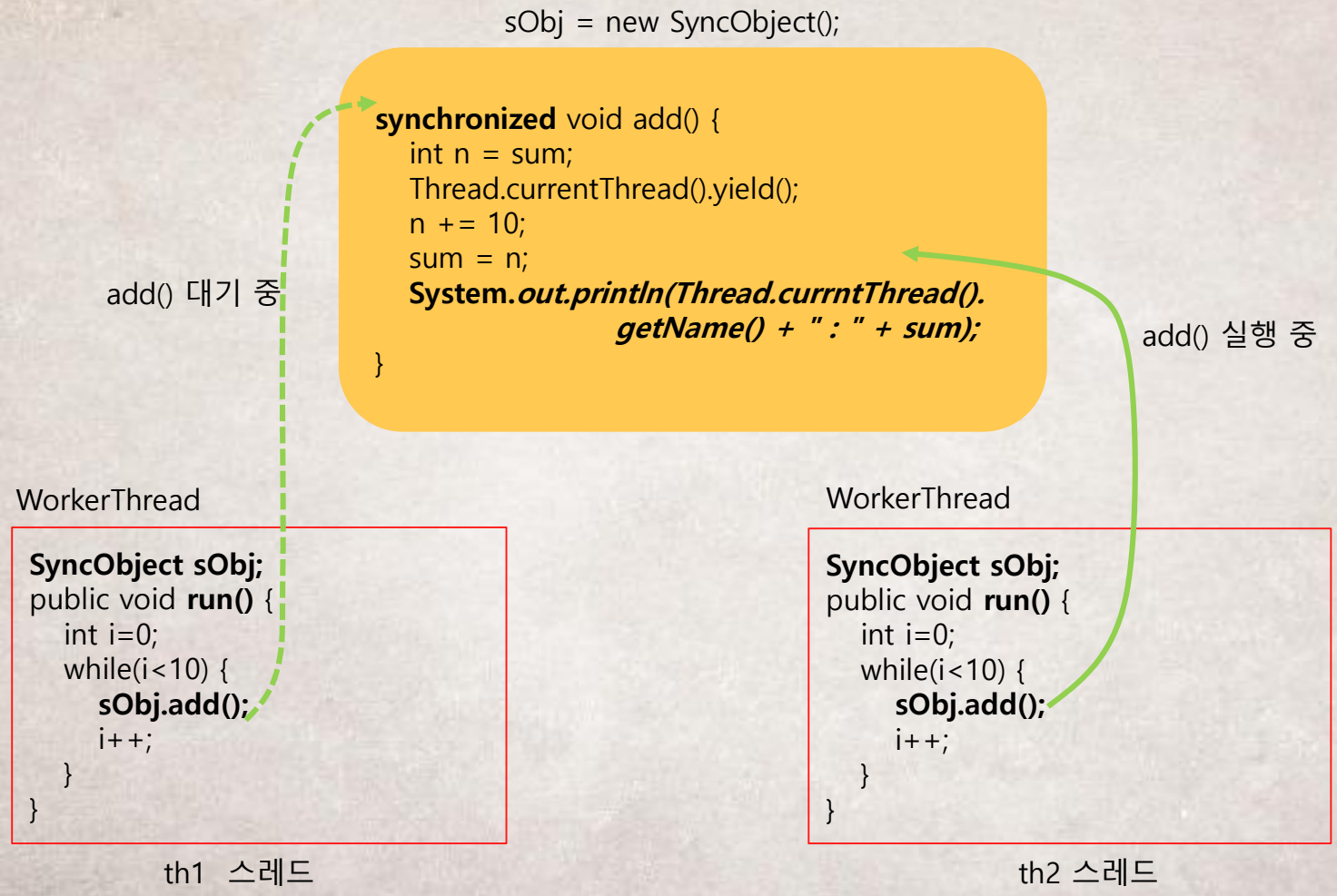
```
1
2  public class WorkerThread extends Thread {
3
4      SyncObject sObj;
5      WorkerThread(String name, SyncObject sObj){
6          super(name);
7          this.sObj = sObj;
8      }
9
10     @Override
11     public void run() {
12         int i = 0;
13
14         while(i < 10){
15             sObj.add();
16             i++;
17         }
18     }
19
20 }
21
```

Problems | Javadoc | Declaration | Console

<terminated> SynchronizedEx [Java Application] C:₩Program Files₩Java₩jre7₩bin₩javaw.exe (2015. 5. 14. 오후 8:51:25)

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

# SyncObject 객체에 대한 스레드의 동시 접근



# 09.4 점수판 예에서 synchronized 사용하지 않을 경우

SyncObject.java | WorkerThread.java | SynchronizedEx.java

```
1
2  class SyncObject {
3      int sum = 0;
4
5      void add(){
6          int n = sum;
7
8          Thread.currentThre
9
10         n += 10;
11         sum = n;
12         System.out.println
13     }
14
15 }
16
```

Problems | Javadoc | Declaration | Console

<terminated> SynchronizedEx [Java Application] C:₩Program Files₩Java₩jre7₩bin₩javaw.exe (2015. 5. 14. 오후 8:56:59)

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

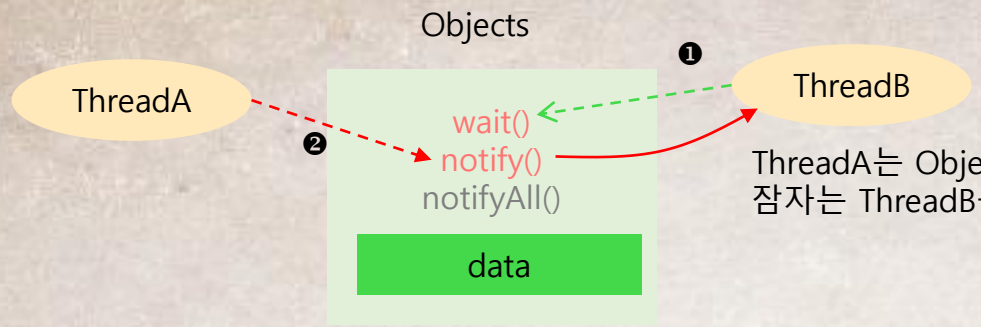
add() 충돌

kitae와 hyosoo가 각각 10번씩 add()를 호출하였지만 동기화가 이루어지지 않아 공유 변수 sum에 대한 접근에 충돌이 있었고, 수를 많이 잃어버리게 되어 누적 점수가 190 밖에 되지 못함

# 동기화 메소드: wait(), notify(), notifyAll()

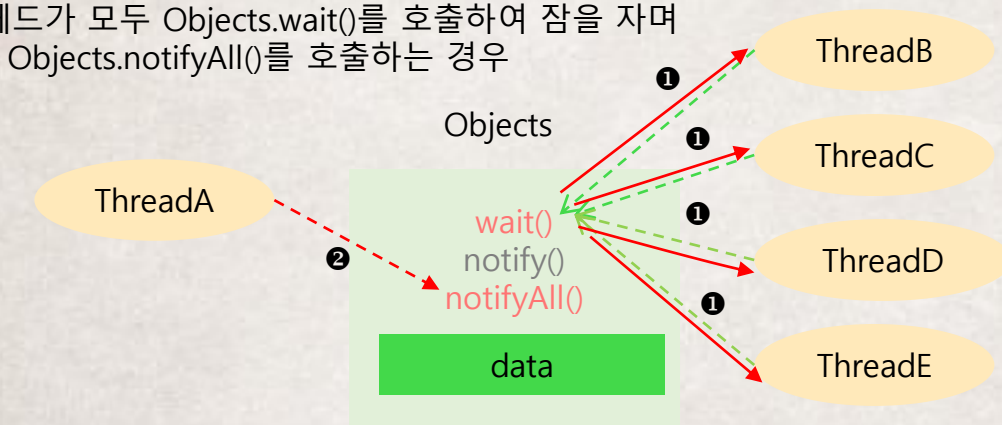
- 동기화 객체
  - 두 개 이상의 스레드 사이에 동기화 작업에 사용되는 객체
- 동기화 메소드
  - synchronized 블럭 내에서만 사용되어야 함
  - wait()
    - 다른 스레드가 notify()를 불러줄 때까지 기다린다.
  - notify()
    - wait() 호출로 인해 대기중인 스레드를 깨우고 RUNNABLE 상태로 한다.
    - 2개 이상의 스레드가 대기중이라도 오직 한 개의 스레드만 깨워 RUNNABLE 상태로 한다.
  - notifyAll()
    - wait() 호출로 인해 대기중인 모든 스레드를 깨우고 이들을 모두 RUNNABLE 상태로 한다.
- 동기화 메소드는 Object 클래스의 메소드이다.
  - 모든 객체가 동기화 객체가 될 수 있다.
  - Thread 객체도 동기화 객체로 사용될 수 있다.

하나의 Thread가 Objects.wait()를 호출하여 잠을 자는 경우



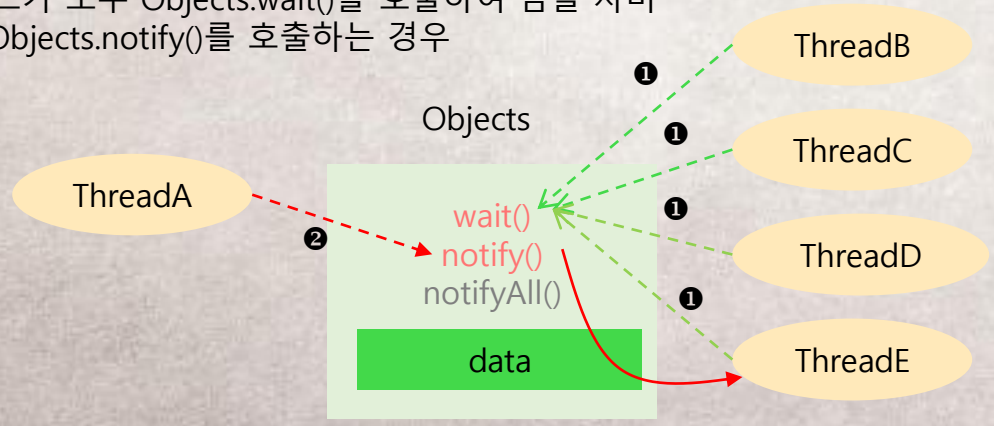

ThreadA는 ObjectS.notify()를 호출하여
잠자는 ThreadB를 깨운다.

4 개의 스레드가 모두 Objects.wait()를 호출하여 잠을 자며
ThreadA는 Objects.notifyAll()를 호출하는 경우




잠자는 4 개의 스레드를
모두 깨운다.

4 개의 스레드가 모두 Objects.wait()를 호출하여 잠을 자며
ThreadA는 Objects.notify()를 호출하는 경우




잠자는 4개의 스레드 중 하나만 깨운다.
선택받지 못한 3 개의 스레드는
계속 잠을 잔다.