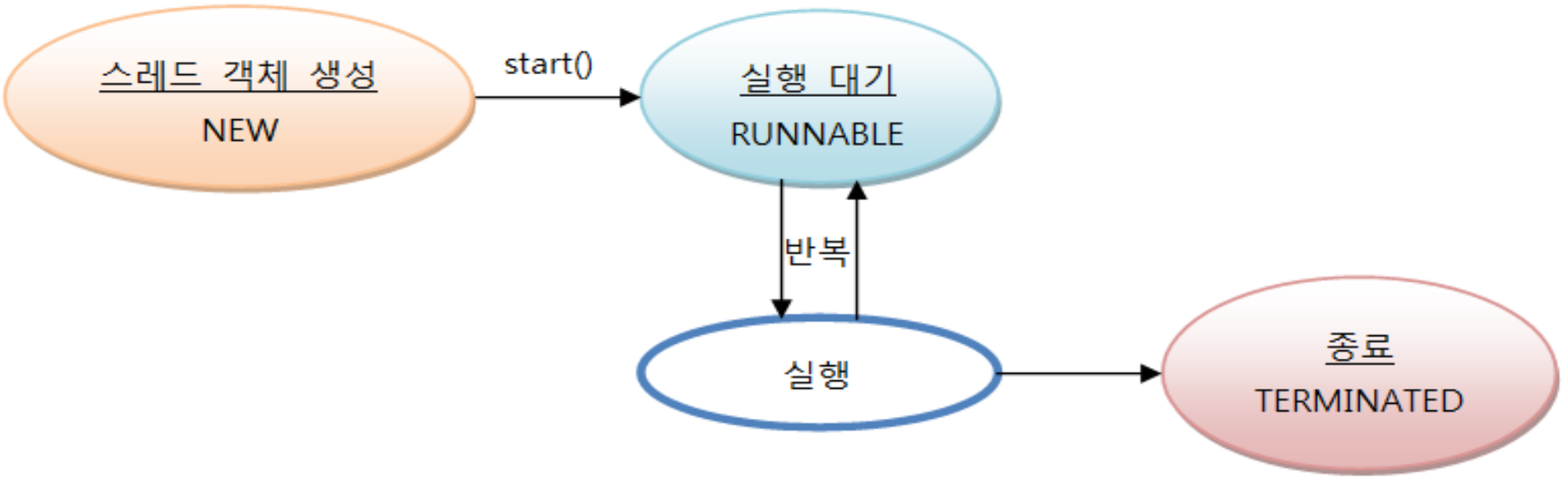
스레드의 상태 제어

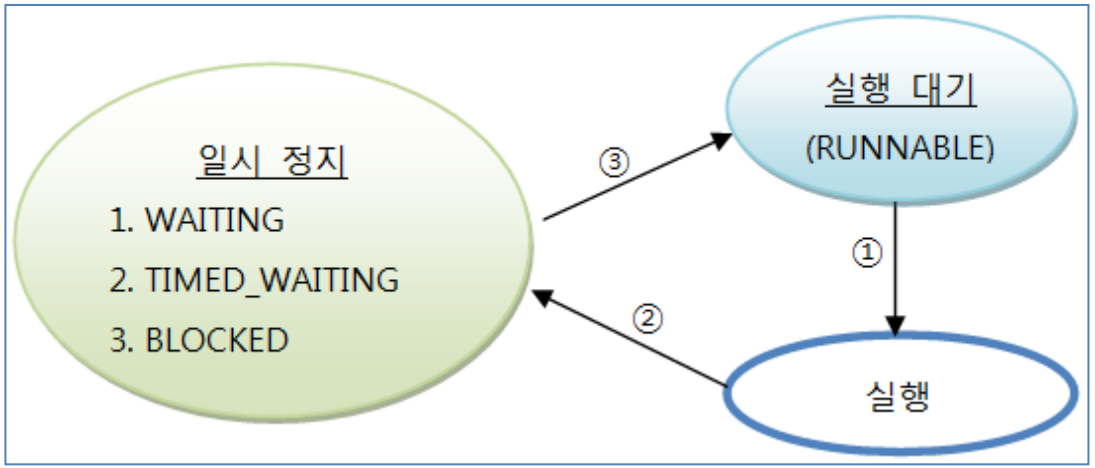
스레드의 상태 제어

❖ 스레드의 일반적인 상태



스레드의 상태 제어

❖ 스레드에 일시 정지 상태 도입한 경우



상태	열거 상수	설명
객체 생성	NEW	스레드 객체가 생성, 아직 start() 메소드가 호출되지 않은 상태
실행 대기	RUNNABLE	실행 상태로 언제든지 갈 수 있는 상태
일시 정지	BLOCKED	사용코저하는 객체의 락이 풀릴 때까지 기다리는 상태
	WAITING	다른 스레드가 통지할 때까지 기다리는 상태
	TIMED_WAITING	주어진 시간 동안 기다리는 상태
종료	TERMINATED	실행을 마친 상태

스레드의 상태 제어

❖ 타겟 스레드의 상태를 출력하는 스레드 : StatePrintThread.java

```
public class StatePrintThread extends Thread {
    private Thread targetThread;

    public StatePrintThread(Thread targetThread) {
        this.targetThread = targetThread;
    }

    public void run() {
        while(true) {
            Thread.State state = targetThread.getState();
            System.out.println("타겟 스레드 상태: " + state);
            if(state == Thread.State.NEW) {
                targetThread.start();
            }
            if(state == Thread.State.TERMINATED) {
                break;
            }
            try {
                //0.5초간 일시 정지
                Thread.sleep(500);
            } catch(Exception e) {}
        }
    }
}
```

스레드의 상태 제어

❖ 타겟 스레드 : TargetThread.java

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

        try {
            //1.5초간 일시 정지
            Thread.sleep(1500);
        } catch(Exception e) {}

        for(long i=0; i<1000000000; i++) {}
    }
}
```

NEW → RUNNABLE → TIMED_WAITING → RUNNABLE → TERMINATED

스레드의 상태 제어

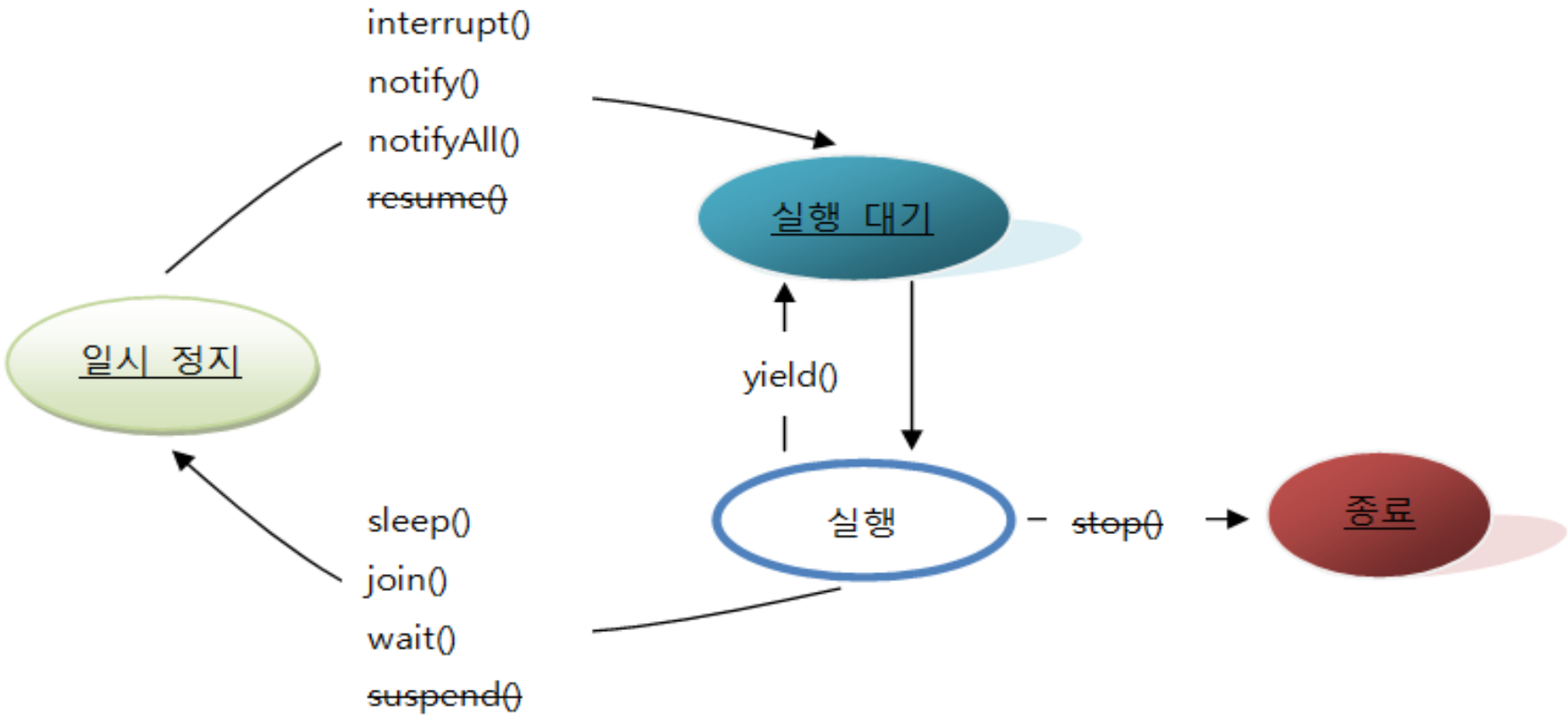
❖ 실행 클래스 : ThreadStateExample.java

```
public class ThreadStateExample {
    public static void main(String[] args) {
        StatePrintThread statePrintThread =
                                new StatePrintThread(new TargetThread());
        statePrintThread.start();
    }
}
```

스레드의 상태 제어

❖ 상태 제어

- 실행 중인 스레드의 상태를 변경하는 것
- 상태 변화를 가져오는 메소드의 종류 (취소선 가진 메소드는 사용 X)



스레드의 상태 제어

❖ 상태 제어

- 상태 변화를 가져오는 메소드의 종류

메소드	설명
interrupt( )	일시 정지 상태의 스레드에서 InterruptedException 예외를 발생시켜, 예외 처리 코드(catch)에서 실행 대기 상태로 가거나 종료 상태로 갈 수 있도록 한다.
notify( ) notifyAll( )	동기화 블록 내에서 wait( ) 메소드에 의해 일시 정지 상태에 있는 스레드를 실행 대기 상태로 만든다.
resume( )	suspend( ) 메소드에 의해 일시 정지 상태에 있는 스레드를 실행 대기 상태로 만든다. - Deprecated (대신 notify( ), notifyAll( ) 사용)
sleep(long millis) sleep(long millis, int nanos)	주어진 시간 동안 스레드를 일시 정지 상태로 만든다. 주어진 시간이 지나면 자동적으로 실행 대기 상태가 된다.
join( ) join(long millis) join(long millis, int nanos)	join( ) 메소드를 호출한 스레드는 일시 정지 상태가 된다. 실행 대기 상태로 가려면, join( ) 메소드를 멤버로 가지는 스레드가 종료되거나, 매개값으로 주어진 시간이 지나야 한다.

스레드의 상태 제어

❖ 상태 제어

- 상태 변화를 가져오는 메소드의 종류

메소드	설명
suspend( )	스레드를 일시 정지 상태로 만든다. resume( ) 메소드를 호출하면 다시 실행 대기 상태가 된다. - Deprecated (대신 wait( ) 사용)
yield( )	실행 중에 우선순위가 동일한 다른 스레드에게 실행을 양보하고 실행 대기 상태가 된다.
stop( )	스레드를 즉시 종료시킨다. - Deprecated

스레드의 상태 제어

❖ 주어진 시간 동안 일시 정지 - sleep()

```
try {
   Thread.sleep(1000);
} catch(InterruptedException  e) {
   // interrupt()  메소드가  호출되면  실행
}
```

- 얼마 동안 일시 정지 상태로 있을 것인지 밀리 세컨드(1/1000) 단위로 지정
- 일시 정지 상태에서 interrupt() 메소드 호출
  - InterruptedException 발생

스레드의 상태 제어

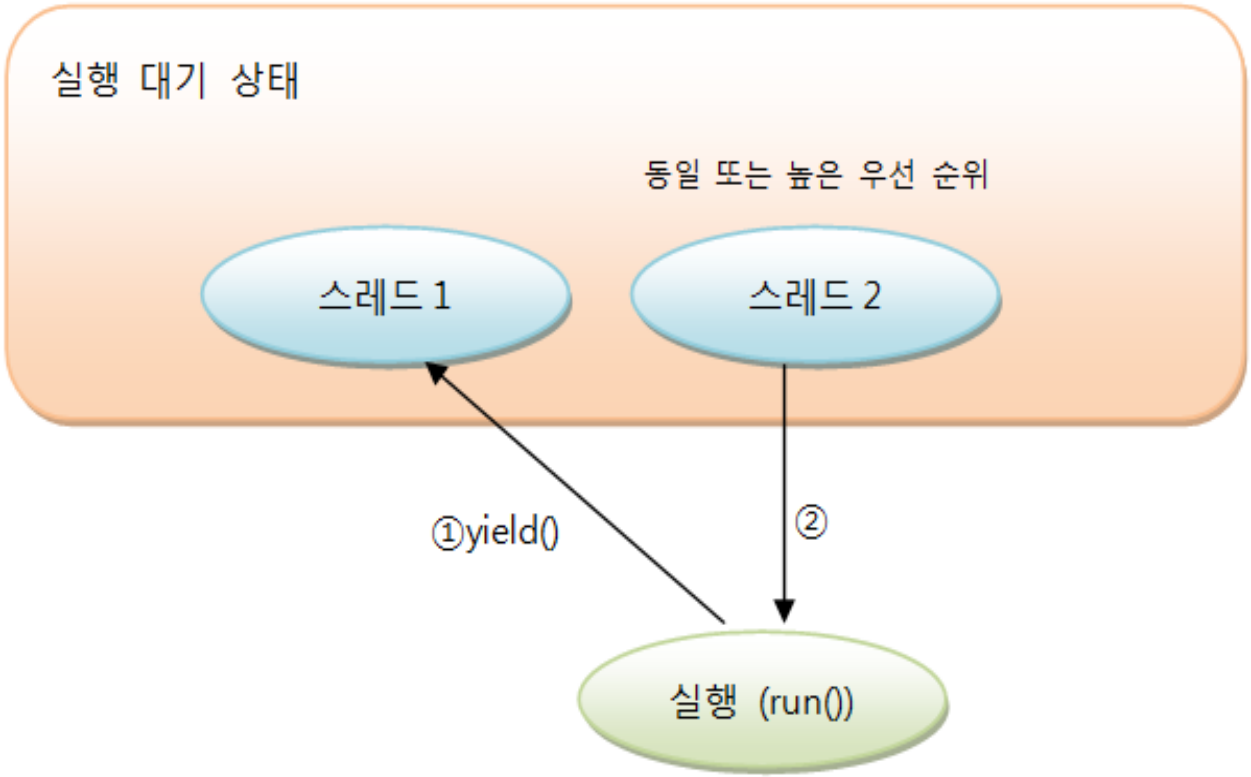
❖ 3초 주기로 10번 비프음 발생 : SleepExample.java

```
import java.awt.Toolkit;

public class SleepExample {
    public static void main(String[] args) {
        Toolkit toolkit = Toolkit.getDefaultToolkit();
        for(int i=0; i<10; i++) {
            toolkit.beep();
            try {
                Thread.sleep(3000);
            } catch(InterruptedException e) {
            }
        }
    }
}
```

스레드의 상태 제어

❖ 다른 스레드에게 실행 양보 - yield()

```
public void run() {
    while(true) {
        if(work) {
            System.out.println("ThreadA 작업 내용");
        } else {
            Thread.yield();
        }
    }
}
```



스레드의 상태 제어

❖ 스레드 실행 양보 예제 : ThreadA.java

```
public class ThreadA extends Thread {
    public boolean stop = false;
    public boolean work = true;

    public void run() {
        while(!stop) {
            if(work) {
                System.out.println("ThreadA 작업 내용");
            } else {
                Thread.yield();
            }
        }
        System.out.println("ThreadA 종료");
    }
}
```

❖ 스레드 실행 양보 예제 : ThreadB.java

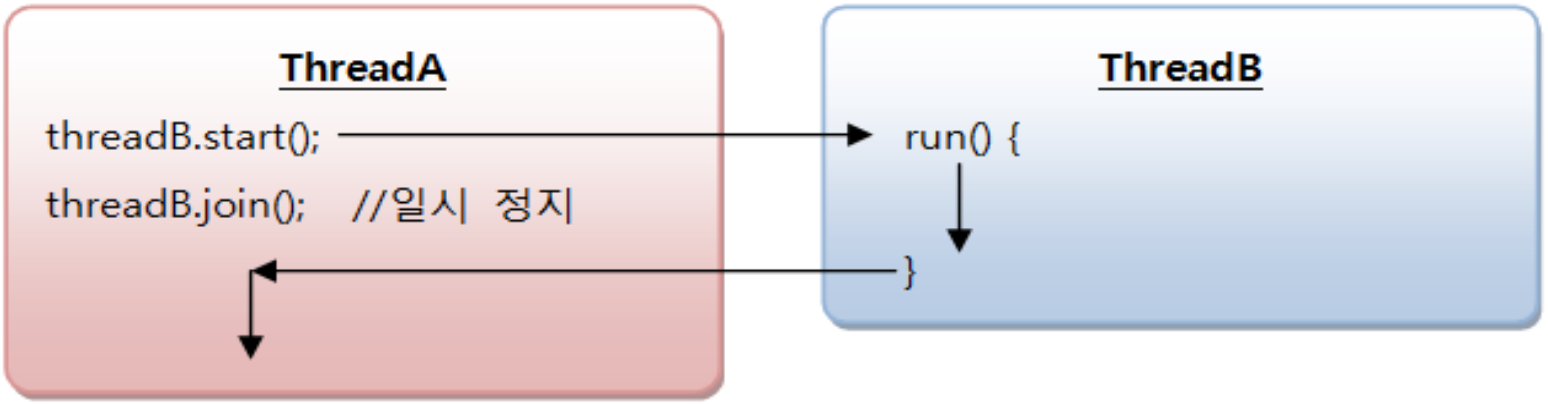
```
public class ThreadB extends Thread {
    public boolean stop = false;
    public boolean work = true;

    public void run() {
        while(!stop) {
            if(work) {
                System.out.println("ThreadB 작업 내용");
            } else {
                Thread.yield();
            }
        }
        System.out.println("ThreadB 종료");
    }
}
```

❖ 스레드 실행 양보 예제 : YieldExample.java

```
public class YieldExample {
    public static void main(String[] args) {
        ThreadA threadA = new ThreadA();
        ThreadB threadB = new ThreadB();
        threadA.start();
        threadB.start();

        try { Thread.sleep(3000); } catch (InterruptedException e) {}
        threadA.work = false;

        try { Thread.sleep(3000); } catch (InterruptedException e) {}
        threadA.work = true;

        try { Thread.sleep(3000); } catch (InterruptedException e) {}
        threadA.stop = true;
        threadB.stop = true;
    }
}
```

스레드의 상태 제어

❖ 다른 스레드의 종료를 기다림 - join()



- 계산 작업을 하는 스레드가 모든 계산 작업 마쳤을 때, 결과값을 받아 이용하는 경우 주로 사용

스레드의 상태 제어

❖ 1부터 100까지 합을 계산하는 스레드 : SumThread.java

```
public class SumThread extends Thread {
    private long sum;

    public long getSum() {
        return sum;
    }

    public void setSum(long sum) {
        this.sum = sum;
    }

    public void run() {
        for(int i=1; i<=100; i++) {
            sum+=i;
        }
    }
}
```

스레드의 상태 제어

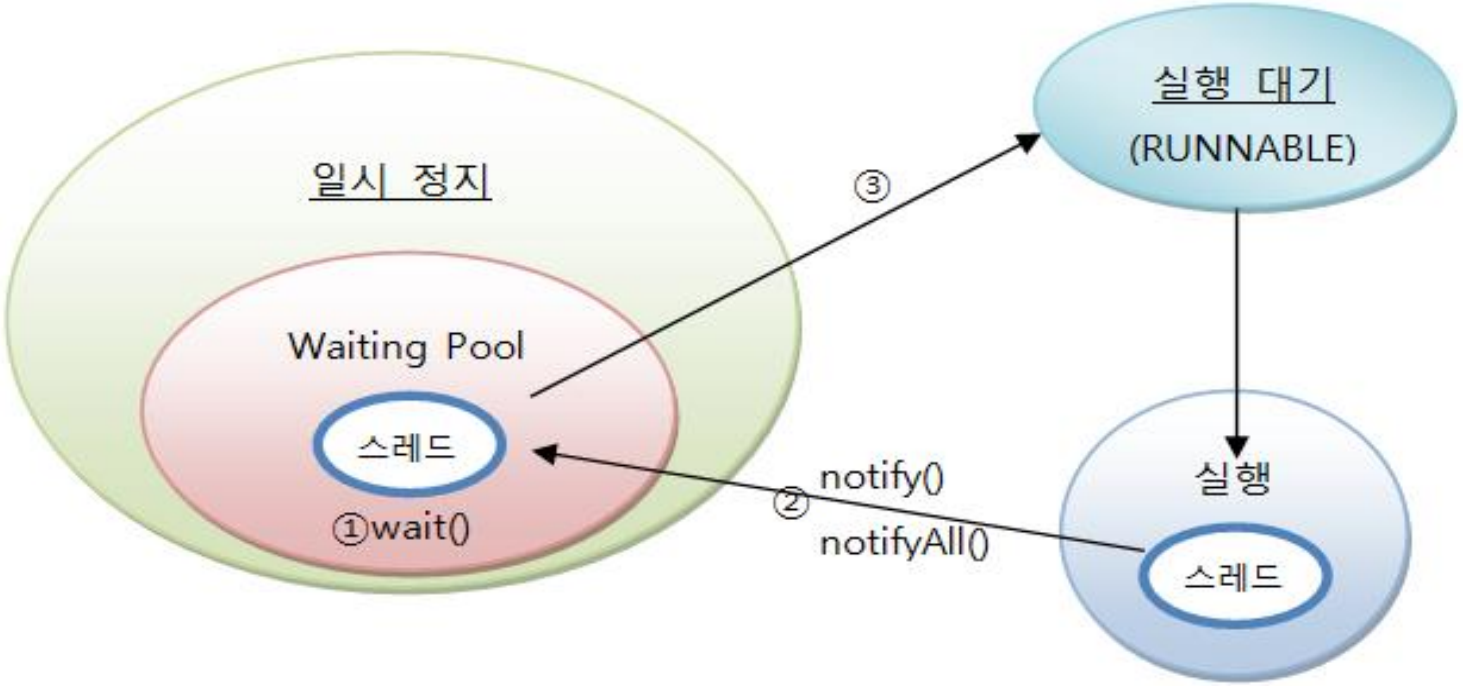
❖ 다른 스레드가 종료될 때까지 일시 정지 상태 유지 : JoinExample.java

```
public class JoinExample {
    public static void main(String[] args) {
        SumThread sumThread = new SumThread();
        sumThread.start();

        try {
            sumThread.join();
        } catch (InterruptedException e) {
        }

        System.out.println("1~100 합: " + sumThread.getSum());
    }
}
```

스레드의 상태 제어

❖ 스레드간 협업 – wait(), notify(), notifyAll()

- 동기화 메소드 또는 블록에서만 호출 가능한 Object의 메소드



스레드의 상태 제어

❖ 두 스레드의 작업 내용을 동기화 메서드로 작성한 공유 객체 : WorkObject.java

```
public class WorkObject {
    public synchronized void methodA() {
        System.out.println("ThreadA의 methodA() 작업 실행");
        notify();    // 일시정지 상태에 있는 ThreadB를 실행 대기 상태로 만듬
        try {
            wait();   // ThreadA를 일시 정지 상태로 만듬
        } catch (InterruptedException e) {
        }
    }

    public synchronized void methodB() {
        System.out.println("ThreadB의 methodB() 작업 실행");
        notify();    // 일시정지 상태에 있는 ThreadA를 실행 대기 상태로 만듬
        try {
            wait();   // ThreadB를 일시 정지 상태로 만듬
        } catch (InterruptedException e) {
        }
    }
}
```

스레드의 상태 제어

❖ ThreadA.java

```
public class ThreadA extends Thread {
    private WorkObject workObject;

    // 공유 객체를 매개값으로 받아 필드에 저장
    public ThreadA(WorkObject workObject) {
        this.workObject = workObject;
    }

    @Override
    public void run() {
        // 공유 객체의 tethodA()를 10번 호출
        for(int i=0; i<10; i++) {
            workObject.methodA();
        }
    }
}
```

❖ WorkObject의 method()를 실행하는 스레드 : ThreadB.java

```
public class ThreadB extends Thread {
    private WorkObject workObject;

    // 공유 객체를 매개값으로 받아 필드에 저장
    public ThreadB(WorkObject workObject) {
        this.workObject = workObject;
    }

    @Override
    public void run() {
        // 공유 객체의 tethodB()를 10번 호출
        for(int i=0; i<10; i++) {
            workObject.methodB();
        }
    }
}
```

스레드의 상태 제어

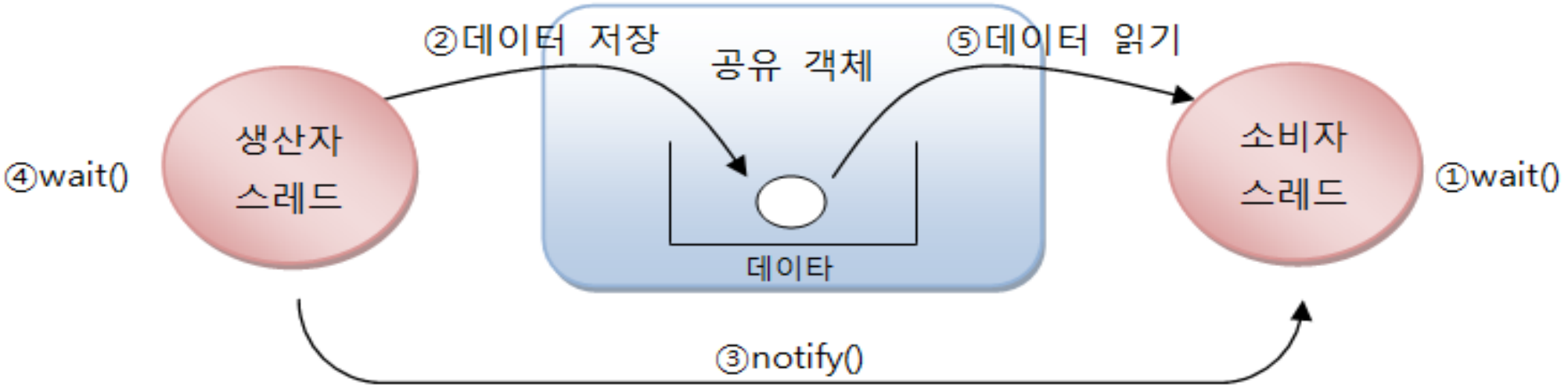
❖ 두 스레드를 생성하고 실행하는 메인 스레드 : WaitNotifyExample.java

```
public class WaitNotifyExample {
    public static void main(String[] args) {
        WorkObject sharedObject = new WorkObject();   // 공유 객체 생성

        ThreadA threadA = new ThreadA(sharedObject);
        ThreadB threadB = new ThreadB(sharedObject);

        threadA.start();
        threadB.start();
    }
}
```

스레드의 상태 제어

❖ 스레드간 협업 – wait(), notify(), notifyAll()

- 두 개의 스레드가 교대로 번갈아 가며 실행해야 할 경우 주로 사용



스레드의 상태 제어

❖ 두 스레드의 작업 내용을 동기화 메서드로 작성한 공유 객체 : DataBox.java

```
public class DataBox {
    private String data;

    public synchronized String getData() {
        // data 필드가 null이면 소비자 스레드를 일시 정지 상태로 만듬
        if(this.data == null) {
            try {
                wait();
            } catch(InterruptedException e) {}
        }

        String returnValue = data;
        System.out.println( " ConsummerThread가 읽은 데이터: " + returnValue);

        // data 필드를 null로 만들고 생산자 스레드를 실행 대기 상태로 만듬
        data = null;
        notify();
        return returnValue;
    }
```

스레드의 상태 제어

❖ 두 스레드의 작업 내용을 동기화 메서드로 작성한 공유 객체 : DataBox.java

```
    public synchronized void setData(String data) {
        // data 필드가 null이 아니면 생산자 스레드를 일시정지 상태로 만듬
        if(this.data != null) {
            try {
                wait();
            } catch(InterruptedException e) {}
        }

        // data 필드에 값을 저장하고 스레드를 실행 대기 상태로 만듬
        this.data = data;
        System.out.println("ProducerThread가 생성한 데이터: " + data);
        notify();
    }
}
```

스레드의 상태 제어

❖ 데이터를 생산(저장)하는 스레드 : ProducerThread.java

```
public class ProducerThread extends Thread {
    private DataBox dataBox;

    public ProducerThread(DataBox dataBox) {
        this.dataBox = dataBox;   // 공유 객체를 필드에 저장
    }

    @Override
    public void run() {
        for(int i=1; i<=3; i++) {
            String data = "Data-" + i;
            dataBox.setData(data);    // 새로운 데이터를 저장
        }
    }
}
```

스레드의 상태 제어

❖ 데이터를 소비하는 (읽는) 스레드 : ConsumerThread.java

```
public class ConsumerThread extends Thread {
    private DataBox dataBox;

    public ConsumerThread(DataBox dataBox) {
        this.dataBox = dataBox;   // 공유 객체를 필드에 저장
    }

    @Override
    public void run() {
        for(int i=1; i<=3; i++) {
            String data = dataBox.getData();    // 새로운 데이터를 읽음
        }
    }
}
```

❖ WaitNotifyExample.java

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

        ProducerThread producerThread = new ProducerThread(dataBox);
        ConsumerThread consumerThread = new ConsumerThread(dataBox);

        producerThread.start();
        consumerThread.start();
    }
}
```

스레드의 상태 제어

❖ 스레드의 안전한 종료 – stop 플래그, interrupt()

- 경우에 따라 실행 중인 스레드 즉시 종료해야 할 필요 있을 때 사용

- stop() 메소드 사용시
  - 스레드 즉시 종료 되는 편리함
  - Deprecated - 사용 중이던 자원들이 불안전한 상태로 남겨짐

스레드의 상태 제어

❖ 안전한 종료 위해 stop 플래그 이용하는 방법

- ㅇ stop 플래그로 메소드의 정상 종료 유도

```
public class XXXThread extends Thread {
  private boolean stop;  //stop 플래그 필드

  public void run() {
    while( !stop ) {  ●----------  stop 이 true 가 되면 run() 이 종료된다.
      스레드가 반복 실행하는 코드;
    }
    //스레드가 사용한 자원 정리
  }
}
```

스레드의 상태 제어

❖ 무한 반복해서 출력하는 스레드 : PrintThread1.java

```
public class PrintThread1 extends Thread {
    private boolean stop;

    public void setStop(boolean stop) {
      this.stop = stop;
    }

    public void run() {
        while(!stop) {
            System.out.println("실행 중");
        }
        System.out.println("자원 정리");
        System.out.println("실행 종료");
    }
}
```

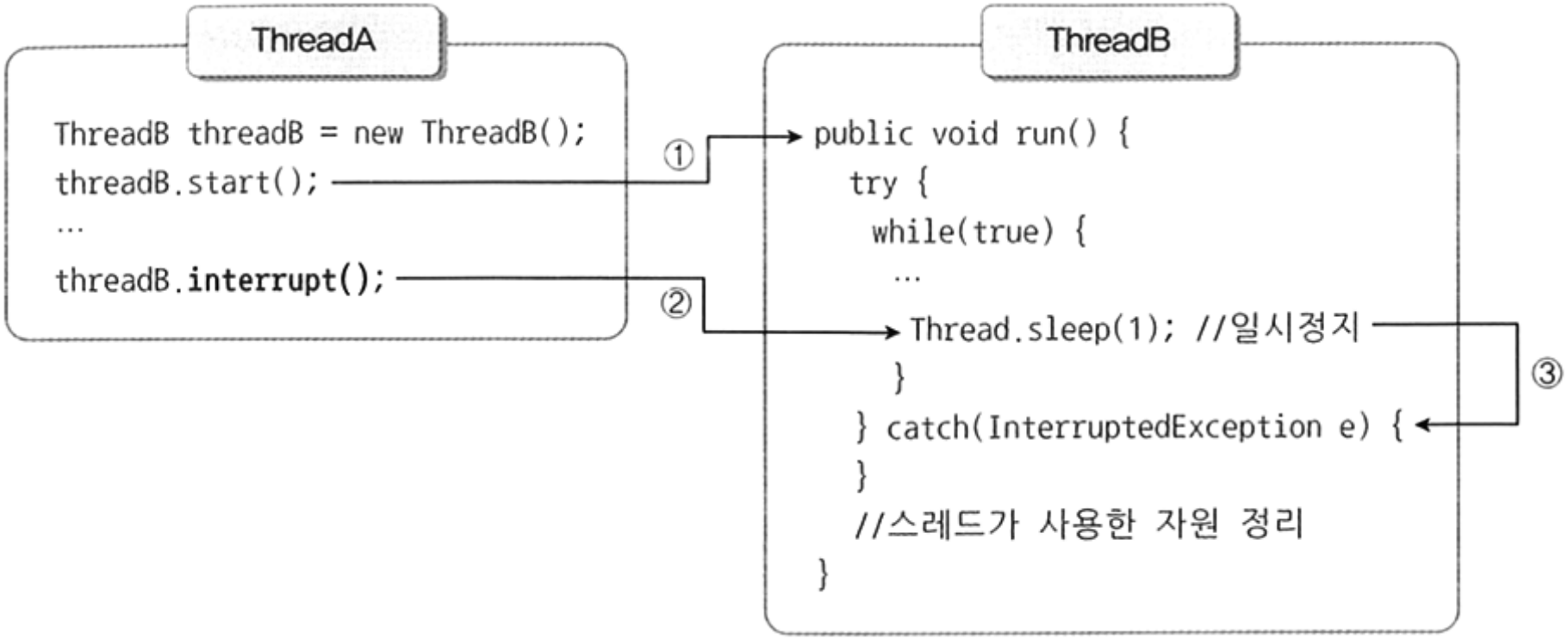
스레드의 상태 제어

❖ 1초 후 출력 스레드를 중지시킴 : StopFlagExample.java

```
public class StopFlagExample {
    public static void main(String[] args)  {
        PrintThread1 printThread = new PrintThread1();
        printThread.start();

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

        // 무한 반복해서 출력하는 스레드
        printThread.setStop(true);
    }
}
```

❖ 스레드의 안전한 종료

- interrupt() 메소드를 이용하는 방법
  - 스레드가 일시 정지 상태일 경우 InterruptedException 발생 시킴
  - 실행대기 또는 실행상태에서는 InterruptedException 발생하지 않음
  - 일시 정지 상태로 만들지 않고 while문 빠져 나오는 방법으로도 쓰임



스레드의 상태 제어

❖ 무한 반복해서 출력하는 스레드 : PrintThread2.java

```
public class PrintThread2 extends Thread {
    public void run() {
        //how1
        /*try {
            while(true) {
                System.out.println("실행 중");
                Thread.sleep(1);
            }
        } catch(InterruptedException e) {
        }*/

        //how2
        while(true) {
            System.out.println("실행 중");
            if(Thread.interrupted()) {
                break;
            }
        }

        System.out.println("자원 정리");
        System.out.println("실행 종료");
    }
}
```

❖ 1초 후 출력 스레드를 중지시킴 : InterruptExample.java

```
public class InterruptExample {
    public static void main(String[] args)  {
        Thread thread = new PrintThread2();
        thread.start();

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

        thread.interrupt();
    }
}
```