

# Java



# 멀티스레드 프로그래밍

멀티스레드 프로그램이란?

멀티스레드 프로그램의 작성 방법

스레드간의 커뮤니케이션 방법

스레드의 상태를 알아내는 방법



# Thread Introduction

## ● MultiTasking

- ▶ Process란 OS에서 실행중인 하나의 Program을 말한다.
- ▶ Multi Process란 두 개 이상의 Process가 실행되는 것을 말한다.
- ▶ MultiTasking이란 두 개 이상의 Process를 실행하여 일을 처리하는 것을 말한다.

## ● Multi Thread

- ▶ Thread란 Process 내에서 실행되는 세부 작업 단위이다.
- ▶ Multi Thread란 하나의 Process에서 여러 개의 Thread가 병행적으로 처리되는 것을 말한다.

# 멀티스레드 프로그래밍

## 01. 멀티스레드 프로그램이란?

### 스레드란?

.. 스레드(thread) : 프로그램의 실행 흐름

```
class Total {  
    public static void main(String args[]) {  
        int total = 0;  
        for (int cnt = 0; cnt < 3; cnt++)  
            total += cnt;  
        System.out.println(total);  
    }  
}
```

프로그램의  
실행 흐름(스레드)



.. 싱글 스레드(single thread program) : 스레드가 하나뿐인 프로그램

.. 멀티스레드 프로그램(multithread program) : 스레드가 둘 이상인 프로그램

# 멀티스레드 프로그래밍

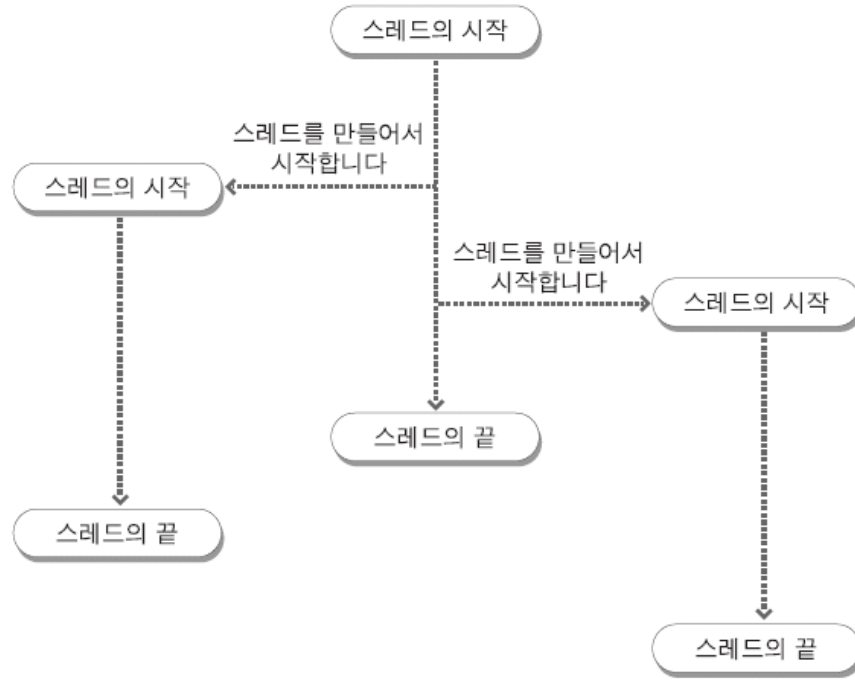
## 01. 멀티스레드 프로그램이란?

### 싱글 스레드/멀티스레드 프로그램

• 작동 방식의 차이



a) 싱글스레드 프로그램의 실행 흐름



b) 멀티스레드 프로그램의 실행 흐름

## ● 02. 멀티스레드 프로그램의 작성 방법

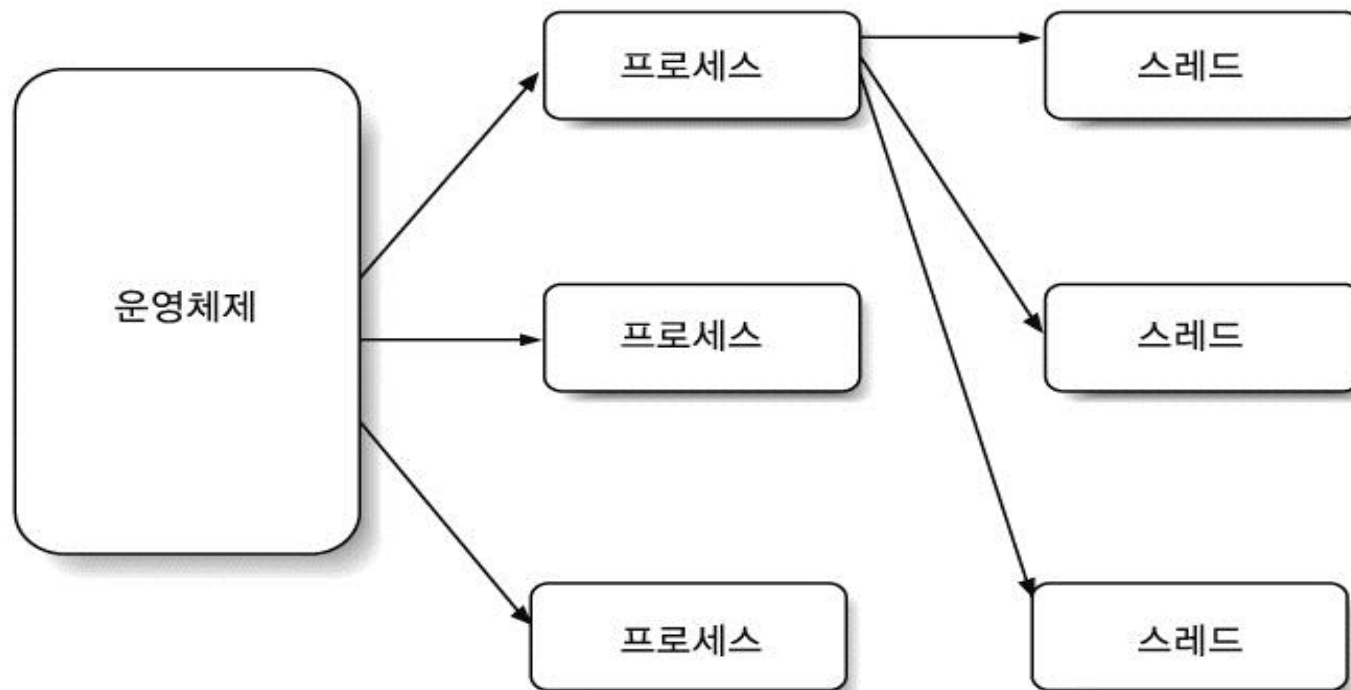
### ● 멀티클래스 프로그램의 작성 방법

•• 다음 두 가지임

- `java.lang.Thread` 클래스를 이용하는 방법
- `java.lang.Runnable` 인터페이스를 이용하는 방법

# Thread Introduction

## ● Process와 Thread의 관계



# Thread State

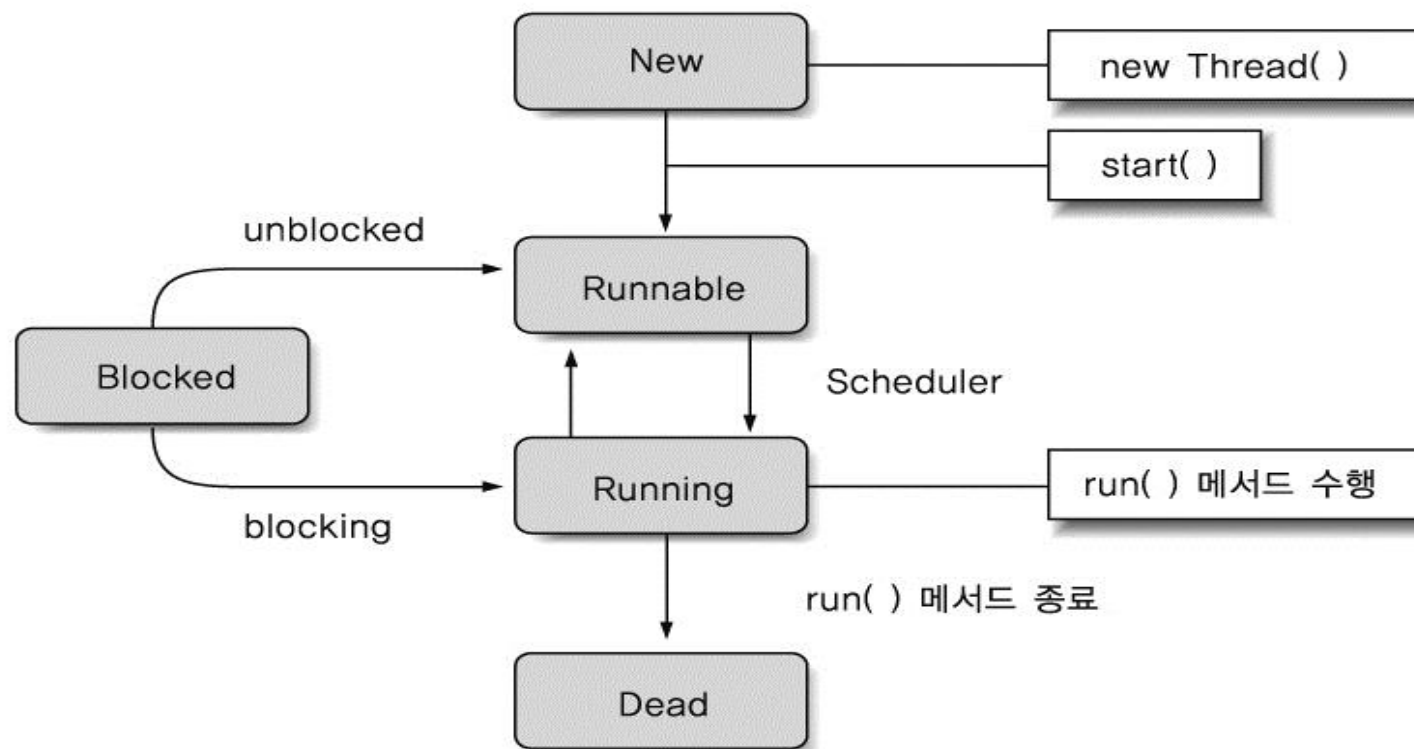
## Thread Life Cycle

- ▶ Thread는 Thread Object가 생성되면 Life Cycle를 갖게 되는데 크게 5가지로 나누게 된다.
- ▶ New – Thread가 만들어진 상태.
- ▶ Runnable – Thread Object가 생성된 후에 start() method를 호출하면 Runnable 상태로 이동하게 된다.
- ▶ Running – Runnable 상태에서 Thread Scheduler에 의해 Running 상태로 이동하게 된다.
- ▶ Blocked – Thread가 다른 특정한 이유로 Running 상태에서 Blocked 상태로 이동하게 된다.
- ▶ Dead – Thread가 종료되면 그 Thread는 다시 시작할 수 없게 된다.



# Thread Introduction

## ● Thread의 Life Cycle

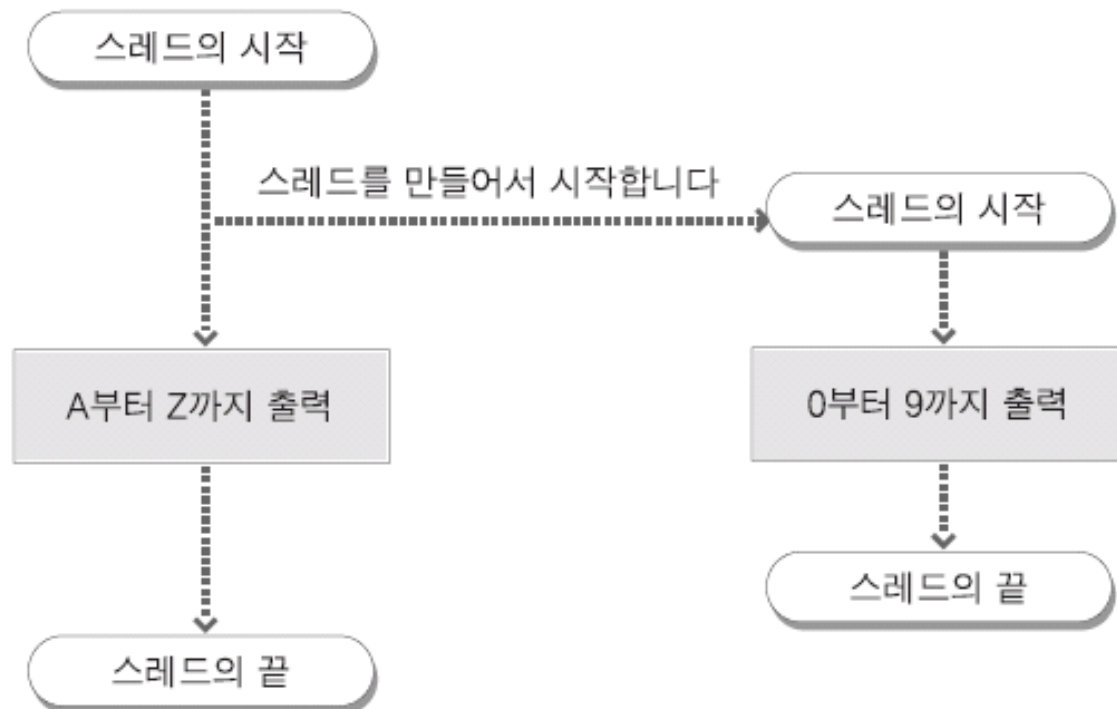


# 멀티스레드 프로그래밍

## 02. 멀티스레드 프로그램의 작성 방법

### Thread 클래스를 이용한 멀티스레드 프로그램

•• 지금부터 작성할 예제의 스레드 구성



# Thread의 생성과 실행

## ● Thread의 생성 방법

- ▶ Thread class를 상속 받는 방법
- ▶ Runnable Interface를 구현하는 방법

## ● Thread class 생성자

※ Thread 클래스의 주요 생성자

생성자	설명
Thread( )	가장 일반적인 형태의 생성자다. 이 생성자를 이용해서 Thread 객체를 생성하게 되면 Thread의 이름은 "Thread-" + n의 형태가 된다.
Thread(Runnable target)	Runnable 객체를 이용해서 Thread 객체를 생성할 수 있는 생성자다.
Thread(Runnable target, String name)	Runnable 객체를 이용해서 Thread 객체를 생성할 수 있는 생성자며, 스레드의 이름을 지정할 수 있는 생성자다.
Thread(String name)	스레드의 이름을 지정하면서 Thread 객체를 생성할 수 있는 생성자다.

# Thread의 생성과 실행

## ● Thread class의 주요 method

※ Thread 클래스의 주요 메서드

반환형	메서드	설명
static void	sleep(long millis)	millis에 지정된 시간만큼 대기한다.
String	getName( )	스레드의 이름을 반환한다.
void	setName(String name)	스레드의 이름을 지정한다.
	start( )	스레드를 시작시킨다.
int	getPriority( )	스레드의 우선순위를 반환한다.
void	setPriority(int newPriority)	스레드의 우선순위를 지정한다.
	join( )	현재 스레드는 join( ) 메서드를 호출한 스레드가 종료할 때까지 기다리게 된다.
static void	yield( )	수행중인 스레드 중 우선순위가 같은 다른 스레드에게 제어권을 넘긴다.
static Thread	currentThread( )	현재 수행되는 스레드 객체를 리턴한다.

# 멀티스레드 프로그래밍

## 02. 멀티스레드 프로그램의 작성 방법

### Thread 클래스를 이용한 멀티스레드 프로그램

- [예제 18-1] 알파벳과 숫자를 동시에 출력하는 멀티스레드 프로그램 (미완성)

main 메소드를 포함하는 클래스

```
1 class MultithreadExample1 {  
2     public static void main(String args[]) {  
3  
4  
5         for (char ch = 'A'; ch <= 'Z'; ch++)  
6             System.out.print(ch);  
7     }  
8 }
```

스레드를 만들어서 시작하는 부분

숫자를 출력하는 스레드 클래스



# 멀티스레드 프로그래밍

## 02. 멀티스레드 프로그램의 작성 방법

### Thread 클래스를 이용한 멀티스레드 프로그램

• 스레드로 실행할 클래스의 선언 방법

- 1) `java.lang.Thread` 클래스를 상속받는 클래스를 선언합니다.

```
class DigitThread extends Thread {  
    ...  
}
```

java.lang.Thread의  
서브클래스로 선언

- \* `java.lang.Thread` 클래스와 서브클래스들을 스레드 클래스(thread class)라고 부름

# Thread의 Scheduling과 우선순위

## Thread Scheduling 방식

- ▶ 선점형 Thread Scheduling 방식은 Thread의 우선권을 가지고 우선순위가 높은 Thread를 먼저 수행시키는 방식이다,
- ▶ 협력형 Thread Scheduler는 실행중인 Thread가 CPU 사용권을 다른 Thread에게 넘길 때까지 기다리는 방식이다,
- ▶ JVM은 우선순위에 따른 선점형 Thread Scheduling 방식을 사용하고 있다,

## Thread Scheduler

- ▶ Multi Thread가 수행될 때 어떤 Thread가 먼저 수행될지는 Thread Scheduler가 결정하게 된다,
- ▶ Java Application에서는 우선순위가 높은 선점형 Thread Scheduler를 사용하고 있다,

# Thread의 Scheduling과 우선순위

## Thread 우선순위

▶ Thread class에서는 Thread의 우선순위를 부여하는 `setPriority(int newPriority)` method를 제공한다.

## Thread class의 우선순위를 정하는 Member 변수

※ Thread 클래스의 우선순위를 정하는 멤버변수

자료형	필드	설명	사용 예
static int	MAX_PRIORITY	스레드가 가질 수 있는 최대 우선순위값(10)	<code>setPriority(Thread.MAX_PRIORITY)</code>
	NORM_PRIORITY	스레드가 가질 수 있는 기본 우선순위값(5)	<code>setPriority(Thread.NORM_PRIORITY)</code>
	MIN_PRIORITY	스레드가 가질 수 있는 최소 우선순위값(1)	<code>setPriority(Thread.MIN_PRIORITY)</code>



## ● 02. 멀티스레드 프로그램의 작성 방법

### ● Thread 클래스를 이용한 멀티스레드 프로그램

#### ● 스레드로 실행할 클래스의 선언 방법

- 2) run 메소드 안에 스레드가 해야할 일을 명령문으로 써넣습니다.

```
class DigitThread extends Thread {  
    public void run() {  
        for (int cnt = 0; cnt < 10; cnt++)  
            System.out.print(cnt);  
    }  
}
```

쓰레드가 해야할 일을  
run 메소드 안에 씁니다

## ● 02. 멀티스레드 프로그램의 작성 방법

### ● Thread 클래스를 이용한 멀티스레드 프로그램

• 스레드를 만들어서 시작하는 방법

- 1) 스레드 클래스의 객체를 생성합니다.

```
Thread thread = new DigitThread();
```

↑  
스레드를 생성하는 식

## 02. 멀티스레드 프로그램의 작성 방법

### Thread 클래스를 이용한 멀티스레드 프로그램

- 스레드를 만들어서 시작하는 방법
  - 2) 스레드 객체에 대해 `start` 메소드를 호출합니다.

```
thread.start();
```

↑  
스레드를 시작하는 메소드

# 멀티스레드 프로그래밍

## 02. 멀티스레드 프로그램의 작성 방법

### Thread 클래스를 이용한 멀티스레드 프로그램

- [예제 18-2] 알파벳과 숫자를 동시에 출력하는 멀티스레드 프로그램 (1)

main 메소드를 포함하는 클래스

```
1 class MultithreadExample1 {
2     public static void main(String args[]) {
3         Thread thread = new DigitThread();    // 스레드를 생성
4         thread.start();                       // 스레드를 시작
5         for (char ch = 'A'; ch <= 'Z'; ch++)
6             System.out.print(ch);
7     }
8 }
```

숫자를 출력하는 스레드 클래스

```
1 class DigitThread extends Thread {
2     public void run() {
3         for (int cnt = 0; cnt < 10; cnt++)
4             System.out.print(cnt);
5     }
6 }
```

명령 프롬프트

```
E:\work\chap18\18-2-1\example1>java MultithreadExample1
ABCDEFGH IJKLMNOPQ0123456789RSTUWXYZ
E:\work\chap18\18-2-1\example1>
```

# 멀티스레드 프로그래밍

## 02. 멀티스레드 프로그램의 작성 방법

### Thread 클래스를 이용한 멀티스레드 프로그램

- java.lang.Thread 클래스의 sleep 메소드

- - 일정 시간이 경과되기를 기다리는 메소드

```
Thread.sleep(1000);
```



주어진 시간이 경과되기를  
기다리는 메소드

## 02. 멀티스레드 프로그램의 작성 방법

### Thread 클래스를 이용한 멀티스레드 프로그램

- java.lang.Thread 클래스의 sleep 메소드
  - InterruptedException의 처리 방법

```
try {  
    Thread.sleep(1000);  
}  
catch (InterruptedException e) {  
    System.out.println(e.getMessage());  
}
```

sleep 메소드가 발생하는  
InterruptedException을  
처리합니다

# 멀티스레드 프로그래밍

## 02. 멀티스레드 프로그램의 작성 방법

### Thread 클래스를 이용한 멀티스레드 프로그램

- [예제 18-3] 알파벳과 숫자를 동시에 출력하는 멀티스레드 프로그램 (2)

main 메소드를 포함하는 클래스

```
1 class MultithreadExample1 {
2     public static void main(String args[]) {
3         Thread thread = new DigitThread();
4         thread.start();
5         for (char ch = 'A'; ch <= 'Z'; ch++) {
6             System.out.print(ch);
7             try {
8                 Thread.sleep(1000);
9             } catch (InterruptedException e) {
10                System.out.println(e.getMessage());
11            }
12        }
13    }
14 }
```

숫자를 출력하는 스레드 클래스

```
1 class DigitThread extends Thread {
2     public void run() {
3         for (int cnt = 0; cnt < 10; cnt++) {
4             System.out.print(cnt);
5             try {
6                 Thread.sleep(1000);
7             } catch (InterruptedException e) {
8                 System.out.println(e.getMessage());
9             }
10        }
11    }
12 }
```



```
C:\> cd E:\work\chap18\18-2-1\example2
E:\work\chap18\18-2-1\example2> java MultithreadExample1
A01B2C3D4E5F6G7H8I9JKLMNOPQRSTUVWXYZ
E:\work\chap18\18-2-1\example2>
```

# 멀티스레드 프로그래밍

## 02. 멀티스레드 프로그램의 작성 방법

### Thread 클래스를 이용한 멀티스레드 프로그램

- [예제 18-4] 네 개의 스레드로 실행되는 다

main 메소드를 포함하는 클래스

```
1 class MultithreadExample2 {
2     public static void main(String args[]) {
3         Thread thread1 = new DigitThread();
4         Thread thread2 = new DigitThread();
5         Thread thread3 = new AlphabetThread();
6         thread1.start();
7         thread2.start();
8         thread3.start();
9     }
10 }
```

} 3 개의 스레드를 생성해서  
시작합니다

명령 프롬프트

```
E:\work\chap18\18-2-1\example3>java MultithreadExample2
00A11B22C33D44E55F66G77H88I99JKLMNOPQRSTUVWXYZ
E:\work\chap18\18-2-1\example3>
```

숫자를 출력하는 스레드 클래스

```
1 class DigitThread extends Thread {
2     public void run() {
3         for (int cnt = 0; cnt < 10; cnt++) {
4             System.out.print(cnt);
5             try {
6                 Thread.sleep(1000);
7             } catch (InterruptedException e) {
8                 System.out.println(e.getMessage());
9             }
10        }
11    }
12 }
```

알파벳을 출력하는 스레드 클래스

```
1 class AlphabetThread extends Thread {
2     public void run() {
3         for (char ch = 'A'; ch <= 'Z'; ch++) {
4             System.out.print(ch);
5             try {
6                 Thread.sleep(500);
7             } catch (InterruptedException e) {
8                 System.out.println(e.getMessage());
9             }
10        }
11    }
12 }
```

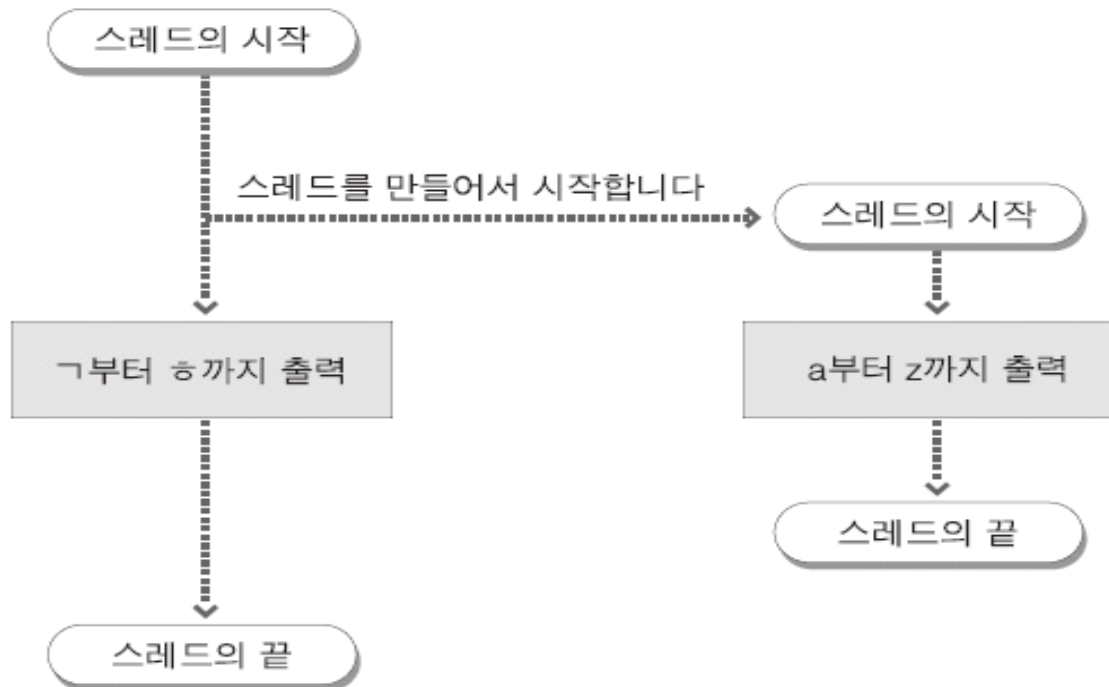


# 멀티스레드 프로그래밍

## 02. 멀티스레드 프로그램의 작성 방법

### Runnable 인터페이스를 이용한 멀티스레드 프로그램

•• 지금부터 작성할 예제의 스레드 구성



# 멀티스레드 프로그래밍

## 02. 멀티스레드 프로그램의 작성 방법

### Runnable 인터페이스를 이용한 멀티스레드 프로그램

- [예제 18-5] 한글과 영문을 동시에 출력하는 멀티스레드 프로그램 (미완성)

main 메소드를 포함하는 클래스

```
1 class MultithreadExample3 {  
2     public static void main(String args[]) {  
3           
4           
5         char arr[] = { 'ㄱ', 'ㄴ', 'ㄷ', 'ㄹ', 'ㅁ', 'ㅂ', 'ㅅ',  
                        'ㅇ', 'ㅈ', 'ㅊ', 'ㅋ', 'ㅌ', 'ㅍ', 'ㅎ' };  
6         for (char ch : arr)  
7             System.out.print(ch);  
8     }  
9 }
```

스레드를 만들어서 시작하는 부분

영문 소문자를 출력하는 스레드 클래스



## 02. 멀티스레드 프로그램의 작성 방법

### Runnable 인터페이스를 이용한 멀티스레드 프로그램

- 스레드로 실행할 클래스의 선언 방법

- 1) `java.lang.Runnable` 인터페이스를 구현하는 클래스를 선언합니다.

```
class SmallLetters implements Runnable {  
    ...  
}
```

`java.lang.Runnable`  
인터페이스를  
구현하는 클래스로 선언

# 멀티스레드 프로그래밍

## 02. 멀티스레드 프로그램의 작성 방법

### Runnable 인터페이스를 이용한 멀티스레드 프로그램

• 스레드로 실행할 클래스의 선언 방법

- 2) run 메소드 안에 스레드가 해야할 일을 명령문으로 써넣습니다.

```
class SmallLetters implements Runnable {  
    public void run() {  
        for (char ch = 'a'; ch <= 'z'; ch++)  
            System.out.print(ch);  
    }  
}
```

스레드가 해야할 일을  
run 메서드 안에 씁니다

주의 : 이런 클래스는 스레드 클래스가 아님

# 멀티스레드 프로그래밍

## 02. 멀티스레드 프로그램의 작성 방법

### Runnable 인터페이스를 이용한 멀티스레드 프로그램

• 스레드를 만들어서 시작하는 방법

- 1) 다음과 같은 방법으로 Thread 객체를 생성합니다.

```
SmallLetters obj = new SmallLetters();
```

Runnable 인터페이스를 구현하는 클래스의 객체를 생성해서 Thread 생성자의 파라미터로 사용합니다.

```
Thread thread = new Thread(obj);
```

# 멀티스레드 프로그래밍

## 02. 멀티스레드 프로그램의 작성 방법

### Runnable 인터페이스를 이용한 멀티스레드 프로그램

- 스레드를 만들어서 시작하는 방법
  - 2) Thread 객체에 대해 start 메소드를 호출합니다.

```
thread.start();
```

↑  
스레드를 시작하는 메소드

# 멀티스레드 프로그래밍

## 02. 멀티스레드 프로그램의 작성 방법

### Runnable 인터페이스를 이용한 멀티스레드 프로그램

- [예제 18-6] 한글과 영문을 동시에 출력하는 멀티스레드 프로그램 (완성)

main 메소드를 포함하는 클래스

```
1 class MultithreadExample3 {
2     public static void main(String args[]) {
3         Thread thread = new Thread(new SmallLetters()); // 스레드를 생성
4         thread.start(); // 스레드를 시작
5         char arr[] = { 'ㄱ', 'ㄴ', 'ㄷ', 'ㄹ', 'ㅁ', 'ㅂ', 'ㅅ',
6                         'ㅇ', 'ㅈ', 'ㅊ', 'ㅋ', 'ㅌ', 'ㅍ', 'ㅎ' };
7         for (char ch : arr)
8             System.out.print(ch);
9     }
}
```

영문 소문자를 출력하는 스레드 클래스

```
1 class SmallLetters implements Runnable {
2     public void run() {
3         for (char ch = 'a'; ch <= 'z'; ch++)
4             System.out.print(ch);
5     }
6 }
```

명령 프롬프트

```
E:\work\chap18\18-2\example1>java MultithreadExample3
ㄱ ㄴ ㄷ ㄹ ㅁ ㅂ ㅅ  ㅇ ㅈ ㅊ ㅋ ㅌ ㅍ ㅎ a b c d e f g h i j k l m n o p q r s t u v w x y z
E:\work\chap18\18-2\example1>
```

## 02. 멀티스레드 프로그램의 작성 방법

### Runnable 인터페이스를 사용해야만 하는 경우

- [예제 18-7] 특정 패키지에 속하는 Numbers 클래스

```
1 package kr.or.kosta; ----- 이 클래스는 kosta.or.kr이라는 도메인을 가진 회사에 속합니다
2 public class Numbers {
3     protected void list(int start, int end) {
4         for (int cnt = start; cnt <= end; cnt++) {
5             System.out.printf("(%d)", cnt);
6         }
7     }
8 }
```

다른 도메인을 가진 회사에서  
이 클래스를 스레드로 활용하려면?



## ● 02. 멀티스레드 프로그램의 작성 방법

### ● Runnable 인터페이스를 사용해야만 하는 경우

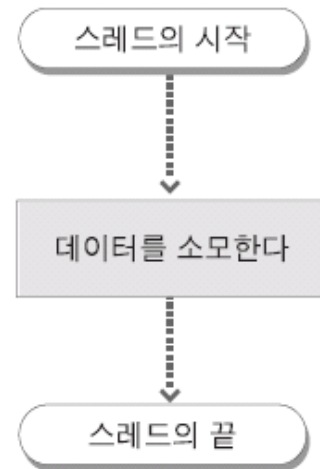
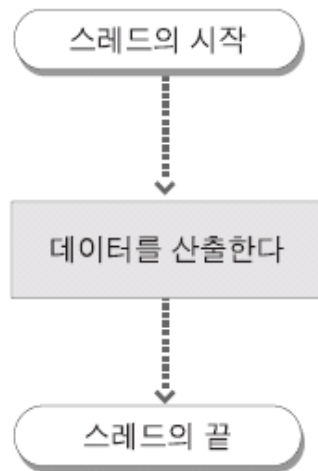
- [예제 18-8] Runnable 인터페이스를 구현하는 Numbers의 서브클래스

```
1  package kr.co.asw; ----- 이 클래스는 asw.co.kr이라는 도메인을 가진 회사에 속합니다
2  public class NumbersRunnable
           extends kr.or.kosta.Numbers implements Runnable {
3      public void run() {
4          list(1, 30); ----- 슈퍼클래스의 메소드 호출
5      }
6  }
```

## 03. 스레드간의 커뮤니케이션

### 스레드간 커뮤니케이션의 필요성

• 다음과 같은 두 스레드가 있다고 가정합니다.

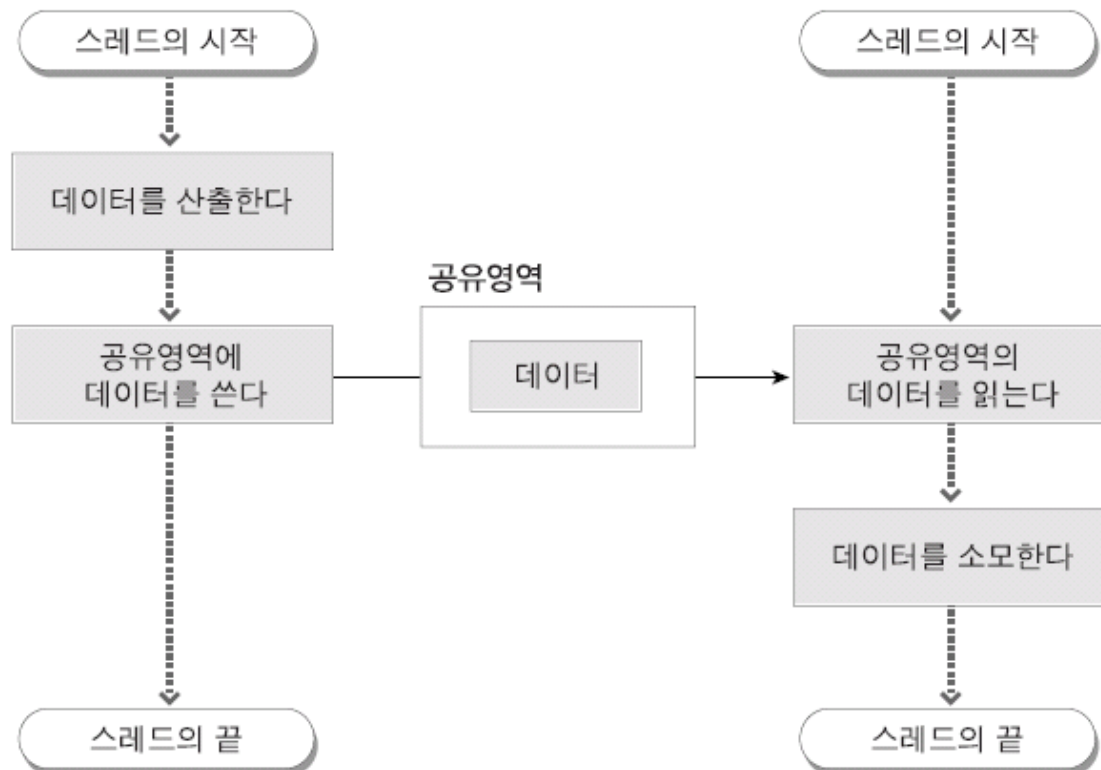


# 멀티스레드 프로그래밍

## 03. 스레드간의 커뮤니케이션

### 스레드간 커뮤니케이션의 필요성

- 두 스레드가 데이터를 교환하는 기본적인 방법

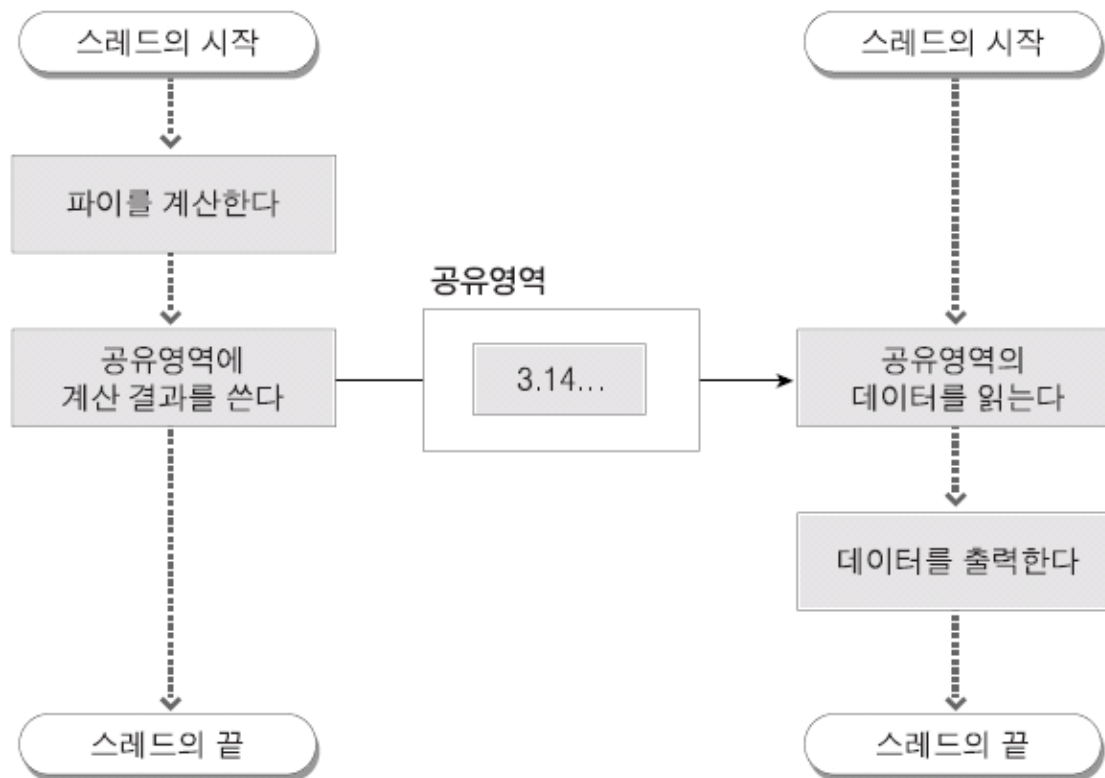


# 멀티스레드 프로그래밍

## 03. 스레드간의 커뮤니케이션

### 스레드간의 데이터 교환

•• 지금부터 작성할 예제의 스레드 구성



# 멀티스레드 프로그래밍

## 03. 스레드간의 커뮤니케이션

### 스레드간의 데이터 교환

#### • 공유 영역을 만드는 방법

- 레퍼런스 타입으로 선언해야 여러 스레드가 참조값을 가지고 접근할 수 있습니다.

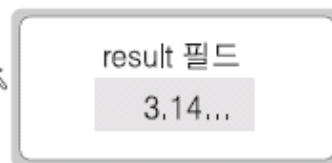
```
class SharedArea {    // 공유 영역을 표현하는 클래스
    double result;
}
```

공유 데이터를  
저장할 필드

파이를 계산하는 스레드



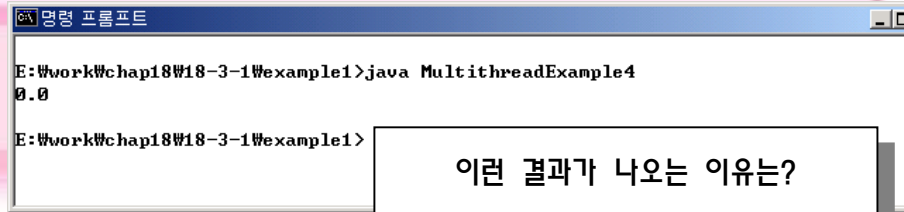
SharedArea 객체



파이를 출력하는 스레드



# 멀티스레드 프로그래밍



## 03. 스레드간의 커뮤니케이션

### 스레드간의 데이터 교환

- [예제 18-9] 원주율  $\pi$  를 계산해서 출력하는 멀티스레드 프로그램 (미완성)

main 메소드를 포함하는 클래스

```
1 class MultithreadExample4 {
2     public static void main(String args[]) {
3         CalcThread thread1 = new CalcThread();
4         PrintThread thread2 = new PrintThread();
5         SharedArea obj = new SharedArea();
6         thread1.sharedArea = obj;
7         thread2.sharedArea = obj;
8         thread1.start();
9         thread2.start();
10    }
11 }
```

공유 영역 객체를 생성해서  
그 객체의 참조값을 두 스레드의  
필드에 저장합니다

공유 영역 클래스

```
1 class SharedArea {
2     double result;
3 }
```

파이를 계산하는 스레드 클래스

```
1 class CalcThread extends Thread {
2     SharedArea sharedArea;
3     public void run() {
4         double total = 0.0;
5         for (int cnt = 1; cnt < 1000000000; cnt += 2)
6             if (cnt / 2 % 2 == 0)
7                 total += 1.0 / cnt;
8             else
9                 total -= 1.0 / cnt;
10        sharedArea.result = total * 4;
11    }
12 }
```

계산 결과를 공유 영역에 씁니다

파일을 출력하는 스레드 클래스

```
1 class PrintThread extends Thread {
2     SharedArea sharedArea;
3     public void run() {
4         System.out.println(sharedArea.result);
5     }
6 }
```

공유 영역의 데이터를 출력합니다

## ● 02. 멀티스레드 프로그램의 작성 방법

### ● 스레드간의 데이터 교환

#### •• 데이터 교환의 타이밍을 맞추는 방법

- 가장 간단한 방법은 공유 영역 안에 데이터 유무를 표시하는 필드를 추가하는 것입니다.

```
class SharedArea {  
    double result;  
    boolean isReady;  
}
```

공유 데이터가 쓰여졌는지 여부를 표현하는 필드

# 멀티스레드 프로그래밍

## 03. 스레드간의 커뮤니케이션

### 스레드간의 데이터 교환

- [예제 18-10] 원주율  $\pi$  를 계산해서 출력하는 멀티스레드 프로그램 (완성)

main 메소드를 포함하는 클래스

```
1 class MultithreadExample4 {
2     public static void main(String args[]) {
3         CalcThread thread1 = new CalcThread();
4         PrintThread thread2 = new PrintThread();
5         SharedArea obj = new SharedArea();
6         thread1.sharedArea = obj;
7         thread2.sharedArea = obj;
8         thread1.start();
9         thread2.start();
10    }
11 }
```

파이를 계산하는 스레드 클래스

```
1 class CalcThread extends Thread {
2     SharedArea sharedArea;
3     public void run() {
4         double total = 0.0;
5         for (int cnt = 1; cnt < 1000000000; cnt += 2)
6             if (cnt / 2 % 2 == 0)
7                 total += 1.0 / cnt;
8             else
9                 total -= 1.0 / cnt;
10        sharedArea.result = total * 4;
11        sharedArea.isReady = true;
12    }
13 }
```

SharedArea 객체의 isReady 필드 값을 true로 설정합니다.

공유 영역 클래스

```
1 class SharedArea {
2     double result;
3     boolean isReady;
4 }
```

공유 데이터가 쓰여졌는지 여부를 표현하는 필드. 디폴트 값은 false



```
E:\work\chap18\18-3-1\example2>java MultithreadExample4
3.141592651589258
E:\work\chap18\18-3-1\example2>
```

파이를 출력하는 스레드 클래스

```
1 class PrintThread extends Thread {
2     SharedArea sharedArea;
3     public void run() {
4         while(sharedArea.isReady != true)
5             continue;
6         System.out.println(sharedArea.result);
7     }
8 }
```

SharedArea 객체의 isReady 필드 값이 true가 될 때까지 루프를 반복합니다.



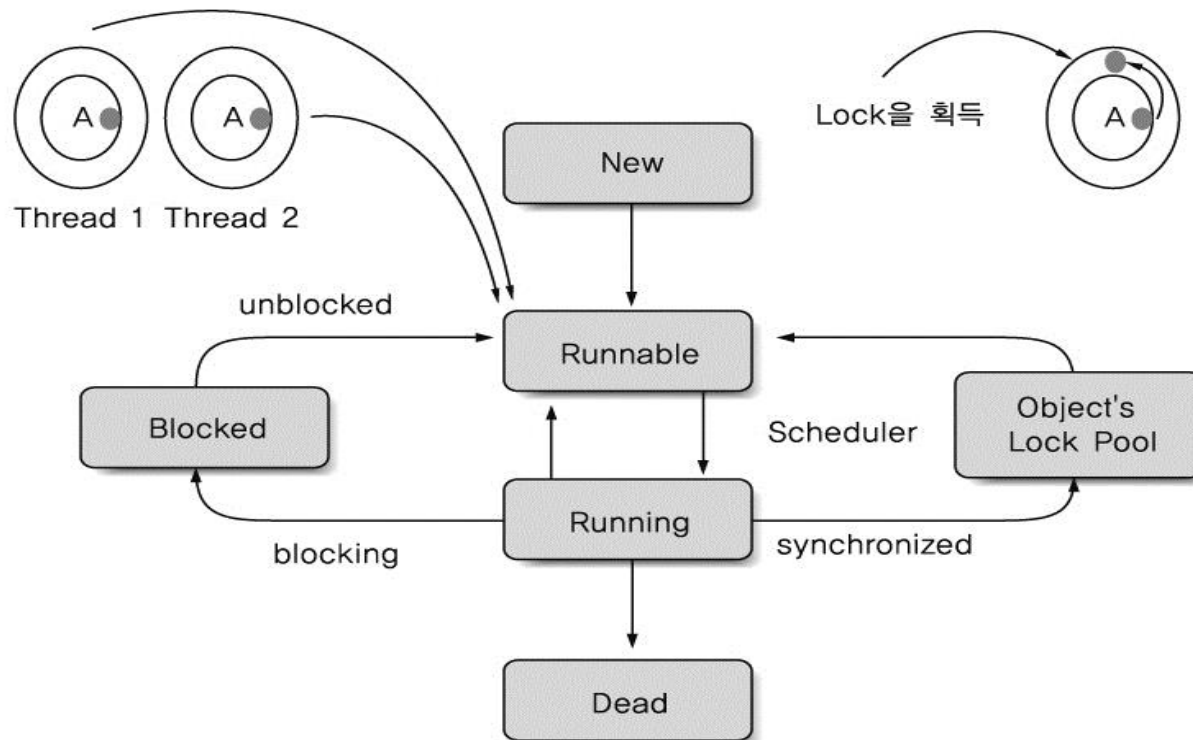
# Synchronization

## ● Synchronization

- ▶ 임계영역이란 Multi Thread에 의해 공유Resource이 참조될 수 있는 code의 범위를 말한다.
- ▶ Multi Thread Program에서 임계영역을 처리하는 경우 심각한 문제가 발생할 수 있다.
- ▶ 이러한 상황을 해결할 수 있는 방법이 Synchronization를 이용하는 것이다.
- ▶ Synchronization를 처리하기 위해 모든 Object에 lock을 포함 시켰다.
- ▶ lock이란 공유 Object에 여러 Thread가 동시에 접근하지 못하도록 하기 위한 것으로 모든 Object가 heap 영역에 생성될 때 자동으로 만들어 진다.

# Synchronization

## ● synchronized 흐름도



# Synchronization

## ● Synchronization 방법

➡ method의 Synchronization 방법

```
public synchronized void synchronizedMethod(){  
    //임계영역 코딩  
}
```

➡ 특정 블록의 Synchronization 방법

```
public void normalMethod(){  
    synchronized(Synchronization할 Object 또는 class명){  
        //임계영역 코딩  
    }  
}
```

# Synchronization

## ● 공정(fairness)

- ▶ 여러 개의 Thread가 하나의 Computing Resource을 사용하기 위해 동시에 접근하는 Program을 작성할 경우 모든 Thread는 공정하게 그 Resource을 사용할 수 있도록 해 주어야 한다.

## ● 기아(starvation)

- ▶ 하나 또는 그 이상의 Thread가 Resource을 얻기 위해 Blocked 상태에 있고, 그 Resource을 얻을 수 없게 되면 다른 작업을 못하는 상태를 말한다.

## ● 교착상태(deadlock)

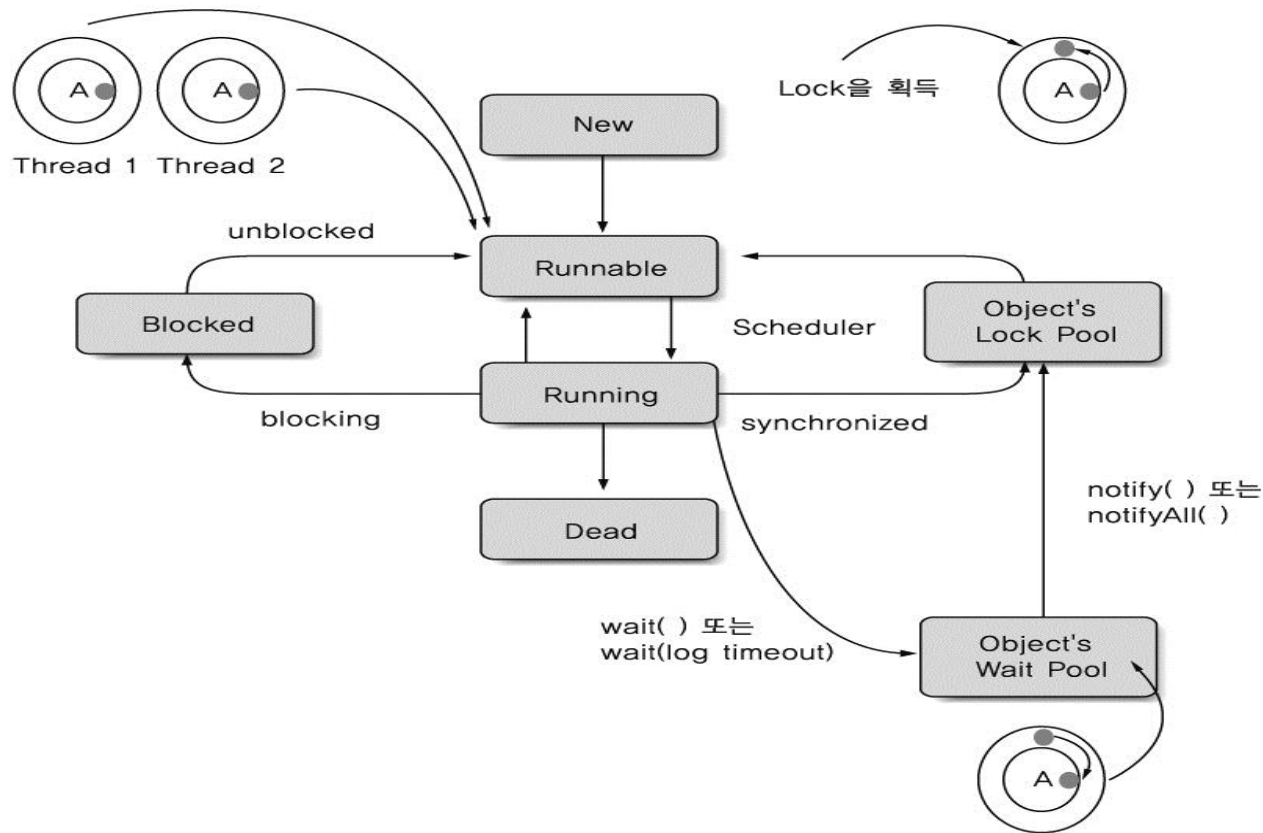
- ▶ 두 개 이상의 Thread가 만족하지 못하는 상태로 계속 기다릴 때 발생한다.

# Synchronization된 생산자와 소비자

- Object class의 `wait()`, `notify()`, `notifyAll()`
  - ➡ Synchronization된 Thread는 Synchronization Block에서 다른 Thread에게 제어권을 넘기지 못한다.
  - ➡ 이와 같이 Synchronization된 Block에서 Thread간의 통신(제어권을 넘김)하기 위해서는 `wait()`, `notify()`, `notifyAll()` method를 사용해야 한다.
  - ➡ 이 method를 사용할 때 주의 해야 할 점은 synchronized Block에서만 의미가 있다.
  - ➡ Synchronized Block이 아닌 경우에 사용할 경우 `java.lang.IllegalMonitorStateException` 이 발생한다.

# Synchronization된 생산자와 소비자

- Object의 `wait()`, `notify()`, `notifyAll()`의 흐름도



## 03. 스레드간의 커뮤니케이션

### critical section의 동기화

#### • critical section

- 스레드 실행 중에 다른 스레드로 제어가 넘어가면 문제를 일으킬 수 있는 부분
- 주로 공유 데이터를 사용하는 부분임

#### • critical section의 동기화(synchronization)

- critical section이 실행되는 동안 다른 스레드가 공유 데이터를 사용할 수 없도록 만드는 것

# 멀티스레드 프로그래밍

## 03. 스레드간의 커뮤니케이션

### critical section의 동기화

- [예제 18-11] 은행 계좌 클래스

```
1  class Account {  
2      String accountNo;    // 계좌번호  
3      String ownerName;    // 예금주 이름  
4      int balance;         // 잔액  
5      Account(String accountNo, String ownerName, int balance) {  
6          this.accountNo = accountNo;  
7          this.ownerName = ownerName;  
8          this.balance = balance;  
9      }  
10     void deposit(int amount) {  
11         balance += amount;  
12     }  
13     int withdraw(int amount) {  
14         if (balance < amount)  
15             return 0;  
16         balance -= amount;  
17         return amount;  
18     }  
19 }
```

지금부터 작성할 예제에서  
사용할 클래스

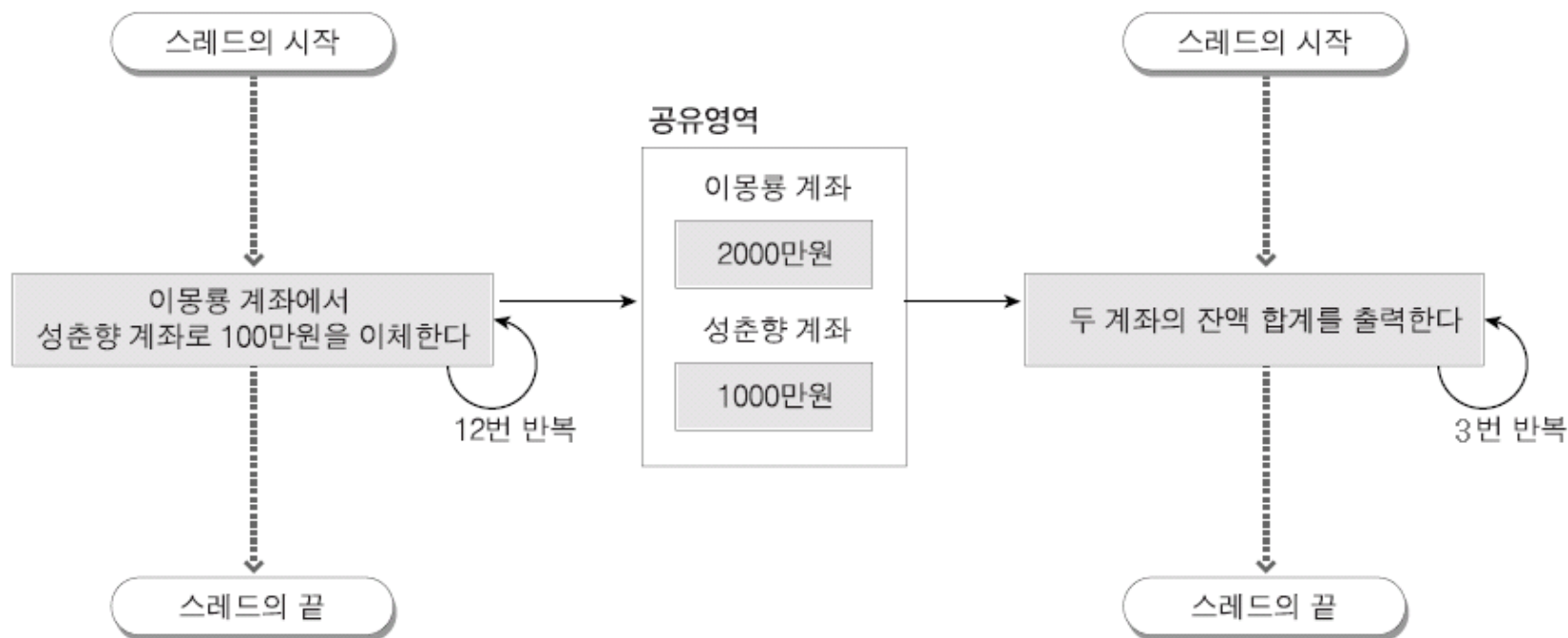


# 멀티스레드 프로그래밍

## 03. 스레드간의 커뮤니케이션

### critical section의 동기화

•• 지금부터 작성할 예제의 스레드 구성



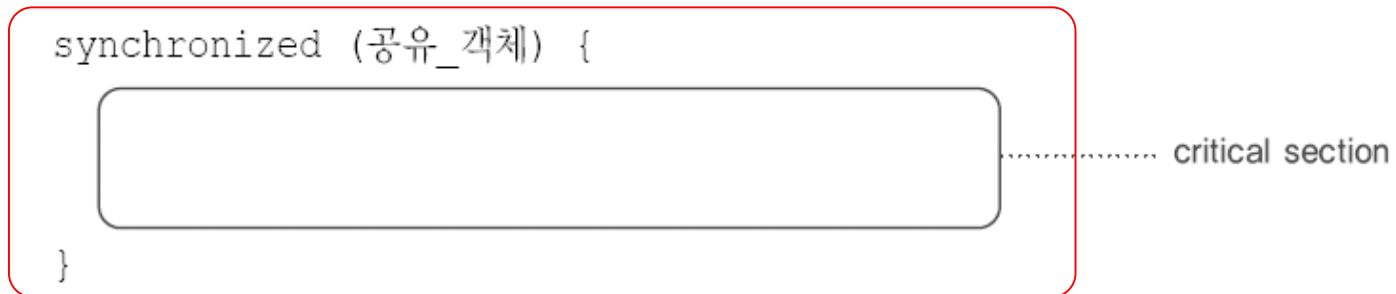
- 03. 스레드간의 커뮤니케이션
- critical section의 동기화

- 50

## 03. 스레드간의 커뮤니케이션

### critical section의 동기화

- critical section의 동기화 방법



동기화 블록(synchronized block)

- [예제 18-13] 계좌 이체와 잔액 합계 출력을 하는 멀티스

```

1  class MultithreadExample5 {
2      public static void main(String args[]) {
3          SharedArea area = new SharedArea();
4          area.account1 = new Account("111-111-1111", "이몽룡", 20000000);
5          area.account2 = new Account("222-222-2222", "성춘향", 10000000);
6          TransferThread thread1 = new TransferThread(area);
7          PrintThread thread2 = new PrintThread(area);
8          thread1.start();
9          thread2.start();
10     }
11 }

```

```

1 class TransferThread extends Thread {
2     SharedArea sharedArea;
3     TransferThread(SharedArea area) {    // 생성자
4         sharedArea = area;
5     }
6     public void run() {
7         for (int cnt = 0; cnt < 12; cnt++) {
8             synchronized (sharedArea) {
9                 sharedArea.account1.withdraw(1000000);
10                System.out.print("이동통 계좌: 100만원 인출,");
11                sharedArea.account2.deposit(1000000);
12                System.out.println("성춘향 계좌: 100만원 입금");
13            }
14        }
15    }
16 }

```

동기화 기술

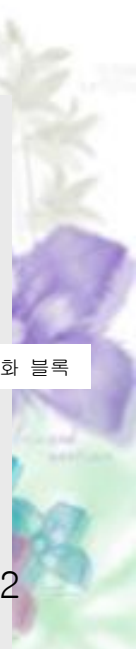
```
1 class SharedArea {
2     Account account1;    // 이몽룡의 계좌
3     Account account2;    // 성춘향의 계좌
4 }
```

```

1  class PrintThread extends Thread {
2      SharedArea sharedArea;
3      PrintThread(SharedArea area) {    // 생성자
4          sharedArea = area;
5      }
6      public void run() {
7          for (int cnt = 0; cnt < 3; cnt++) {
8              synchronized (sharedArea) {
9                  int sum = sharedArea.account1.balance +
10                     sharedArea.account2.balance;
11                  System.out.println("계좌 잔액 합계: " + sum);
12              }
13              try {
14                  Thread.sleep(1);
15              } catch (InterruptedException e) {
16                  System.out.println(e.getMessage());
17              }
18          }
19      }
20  }

```

동기화 기법



# 멀티스레드 프로그래밍

## 03. 스레드간의 커뮤니케이션 critical section의 동기화

[예제 18-13]을 이렇게  
고칠 수도 있습니다.

- [예제 18-14] 계좌 이체와 잔액 합계 출력을 하는 멀티스레드 프로그램(2)

main 메소드를 포함하는 클래스

```
1 class MultithreadExample6 {
2     public static void main(String args[]) {
3         SharedArea area = new SharedArea();
4         area.account1 = new Account("111-111-1111", "이몽룡", 20000000);
5         area.account2 = new Account("222-222-2222", "성춘향", 10000000);
6         TransferThread thread1 = new TransferThread(area);
7         PrintThread thread2 = new PrintThread(area);
8         thread1.start();
9         thread2.start();
10    }
11 }
```

공유 영역 클래스

```
1 class SharedArea {
2     Account account1; // 이몽룡의 계좌
3     Account account2; // 성춘향의 계좌
4     void transfer(int amount) { // 계좌 이체를 한다
5         synchronized (this) {
6             account1.withdraw(1000000);
7             System.out.print("이몽룡 계좌: 100만원 인출,");
8             account2.deposit(1000000);
9             System.out.println("성춘향 계좌: 100만원 입금");
10        }
11    }
12    int getTotal() { // 잔액 합계를 구한다
13        synchronized (this) {
14            return account1.balance + account2.balance;
15        }
16    }
17 }
```

[계좌 이체를 수행하는 스레드 클래스

```
1 class TransferThread extends Thread {
2     SharedArea sharedArea;
3     TransferThread(SharedArea area) { // 생성자
4         sharedArea = area;
5     }
6     public void run() {
7         for (int cnt = 0; cnt < 12; cnt++) {
8             sharedArea.transfer(100); ----- 계좌 이체 메소드 호출
9         }
10    }
11 }
```

[계좌 잔액 합계를 출력하는 스레드 클래스

```
1 class PrintThread extends Thread {
2     SharedArea sharedArea;
3     PrintThread(SharedArea area) { // 생성자
4         sharedArea = area;
5     }
6     public void run() {
7         for (int cnt = 0; cnt < 3; cnt++) {
8             int sum = sharedArea.getTotal(); ----- 잔액 합계 메소드 호출
9             System.out.println("계좌 잔액 합계: " + sum);
10            try {
11                Thread.sleep(1);
12            } catch (InterruptedException e) {
13                System.out.println(e.getMessage());
14            }
15        }
16    }
17 }
```

## 03. 스레드간의 커뮤니케이션

### 동기화 메소드

#### •• 메소드를 동기화하는 방법

- - 메소드 선언 제일 앞에 `synchronized` 키워드를 쓰면 됩니다.

```
synchronized void transfer(int amount) {  
    account1.withdraw(1000000);  
    System.out.print("이몽룡 계좌: 100만원 인출,");  
    account2.deposit(1000000);  
    System.out.println("성춘향 계좌: 100만원 입금");  
}
```

동기화 메소드

### 03. 스레드간의 커뮤니케이션

#### 동기화 메소드

```

E:\work\chap18\18-3-2\example4>java MultithreadExample6
이 줄은 계좌: 100만원 인 줄은 계좌: 100만원 인 줄
계좌잔액 합계: 30000000
이 줄은 계좌: 100만원 인 줄은 계좌: 100만원 인 줄
이 줄은 계좌: 100만원 인 줄은 계좌: 100만원 인 줄
계좌잔액 합계: 30000000
이 줄은 계좌: 100만원 인 줄은 계좌: 100만원 인 줄
이 줄은 계좌: 100만원 인 줄은 계좌: 100만원 인 줄
이 줄은 계좌: 100만원 인 줄은 계좌: 100만원 인 줄
계좌잔액 합계: 30000000
이 줄은 계좌: 100만원 인 줄은 계좌: 100만원 인 줄
이 줄은 계좌: 100만원 인 줄은 계좌: 100만원 인 줄
이 줄은 계좌: 100만원 인 줄은 계좌: 100만원 인 줄
이 줄은 계좌: 100만원 인 줄은 계좌: 100만원 인 줄
이 줄은 계좌: 100만원 인 줄은 계좌: 100만원 인 줄

```

```

1  class MultithreadExample6 {
2      public static void main(String args[]) {
3          SharedArea area = new SharedArea();
4          area.account1 = new Account("111-111-1111", "이몽룡", 20000000);
5          area.account2 = new Account("222-222-2222", "성춘향", 10000000);
6          TransferThread thread1 = new TransferThread(area);
7          PrintThread thread2 = new PrintThread(area);
8          thread1.start();
9          thread2.start();
10     }
11 }

```

```

1  class SharedArea {
2      Account account1;    // 이몽룡의 계좌
3      Account account2;    // 성춘향의 계좌
4      synchronized void transfer(int amount) {
5          account1.withdraw(1000000);
6          System.out.print("이몽룡 계좌: 100만원 인출,");
7          account2.deposit(1000000);
8          System.out.println("성춘향 계좌: 100만원 입금");
9      }
10     synchronized int getTotal() {
11         return account1.balance + account2.balance;
12     }
13 }

```

} 동기화 메소드

} 동기화 메소드

```

1  class TransferThread extends Thread {
2      SharedArea sharedArea;
3      TransferThread(SharedArea area) {
4          sharedArea = area;
5      }
6      public void run() {
7          for (int cnt = 0; cnt < 12; cnt++) {
8              sharedArea.transfer(100);
9          }
10     }
11 }

```

```

1  class PrintThread extends Thread {
2      SharedArea sharedArea;
3      PrintThread(SharedArea area) {
4          sharedArea = area;
5      }
6      public void run() {
7          for (int cnt = 0; cnt < 3; cnt++) {
8              int sum = sharedArea.getTotal();
9              System.out.println("계좌 잔액 합계: " + sum);
10             try {
11                 Thread.sleep(1);
12             } catch (InterruptedException e) {
13                 System.out.println(e.getMessage());
14             }
15         }
16     }
17 }

```

## 03. 스레드간의 커뮤니케이션

### 스레드간의 신호 전송

- 신호를 보내는 `notify` 메소드와 신호를 받는 `wait` 메소드의 호출 방법

`obj.notify();`

다른 스레드로 신호를  
보내는 메소드

`obj.wait();`

다른 스레드로부터  
신호가 오기를 기다리는 메소드

같은 객체여야만 신호를 주고 받을 수 있습니다.



## 03. 스레드간의 커뮤니케이션

### 스레드간의 신호 전송

- 신호를 주고 받는 `notify` 메소드와 `wait` 메소드

```
class CalcThread extends Thread {  
    ...  
    public void run() {  
        obj.notify();  
    }  
}
```

신호를 보냅니다

```
class PrintThread extends Thread {  
    ...  
    public void run() {  
        obj.wait();  
    }  
}
```

# 멀티스레드 프로그래밍

## 03. 스레드간의 커뮤니케이션

### 스레드간의 신호 전송

- [예제 18-16] notify, wait 메소드의 사용 예를 보여주는 원주율 계산 프로그램

main 메소드를 포함하는 클래스

```
1 class MultithreadExample7 {
2     public static void main(String args[]) {
3         CalcThread thread1 = new CalcThread();
4         PrintThread thread2 = new PrintThread();
5         SharedArea obj = new SharedArea();
6         thread1.sharedArea = obj;
7         thread2.sharedArea = obj;
8         thread1.start();
9         thread2.start();
10    }
11 }
```

파이를 계산하는 스레드 클래스

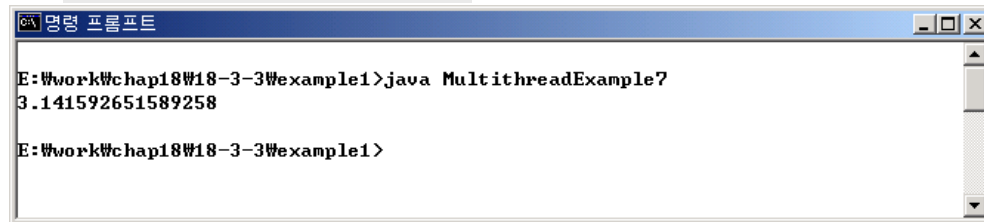
```
1 class CalcThread extends Thread {
2     SharedArea sharedArea;
3     public void run() {
4         double total = 0.0;
5         for (int cnt = 1; cnt < 1000000000; cnt += 2)
6             if (cnt / 2 % 2 == 0)
7                 total += 1.0 / cnt;
8             else
9                 total -= 1.0 / cnt;
10        sharedArea.result = total * 4;
11        sharedArea.isReady = true;
12        synchronized (sharedArea) {
13            sharedArea.notify(); ----- 신호를 보냅니다.
14        }
15    }
16 }
```

공유 영역 클래스

```
1 class SharedArea {
2     double result;
3     boolean isReady;
4 }
```

파일을 출력하는 스레드 클래스

```
1 class PrintThread extends Thread {
2     SharedArea sharedArea;
3     public void run() {
4         if (sharedArea.isReady != true) {
5             try {
6                 synchronized (sharedArea) {
7                     sharedArea.wait(); ----- 신호를 받습니다.
8                 }
9             }
10            catch (InterruptedException e) {
11                System.out.println(e.getMessage());
12            }
13        }
14        System.out.println(sharedArea.result);
15    }
16 }
```



## 03. 스레드간의 커뮤니케이션

### 스레드간의 신호 전송

- 대기하고 있는 모든 스레드로 신호를 보내는 `notifyAll` 메소드

`obj.notifyAll();`

↑  
wait하고 있는 모든 스레드에게  
신호를 보내는 메소드

# 멀티스레드 프로그래밍

## 03. 스레드간의 커뮤니케이션

### 스레드간의 신호 전송

- [예제 18-17] notifyAll 메소드의 사용 예

```
명령 프롬프트
E:\work\chap18\18-3-3\example2>java MultithreadExample8
3.141592651589258
3.14
***  $\pi$  = 3.141592651589258 ***
E:\work\chap18\18-3-3\example2>
```

main 메소드를 포함하는 클래스

```
1 class MultithreadExample8 {
2     public static void main(String args[]) {
3         CalcThread thread1 = new CalcThread();
4         PrintThread thread2 = new PrintThread();
5         SimplePrintThread thread3 = new SimplePrintThread();
6         LuxuryPrintThread thread4 = new LuxuryPrintThread();
7         SharedArea obj = new SharedArea();
8         thread1.sharedArea = obj;
9         thread2.sharedArea = obj;
10        thread3.sharedArea = obj;
11        thread4.sharedArea = obj;
12        thread1.start();
13        thread2.start();
14        thread3.start();
15        thread4.start();
16    }
17 }
```

공유 영역 클래스

```
1 class SharedArea {
2     double result;
3     boolean isReady;
4 }
```

파일을 예쁘게 출력하는 스레드 클래스

```
1 class LuxuryPrintThread extends Thread {
2     SharedArea sharedArea;
3     public void run() {
4         if (sharedArea.isReady != true) {
5             synchronized (sharedArea) {
6                 try {
7                     sharedArea.wait(); -- 신호를 기다립니다.
8                 }
9                 catch (InterruptedException e) {
10                    System.out.println(e.getMessage());
11                }
12            }
13        }
14        System.out.println("***  $\pi$  = " + sharedArea.result + " ***");
15    }
16 }
```

파일을 계산하는 스레드 클래스

```
1 class CalcThread extends Thread {
2     SharedArea sharedArea;
3     public void run() {
4         double total = 0.0;
5         for (int cnt = 1; cnt < 1000000000; cnt += 2)
6             if (cnt / 2 % 2 == 0)
7                 total += 1.0 / cnt;
8             else
9                 total -= 1.0 / cnt;
10        sharedArea.result = total * 4;
11        sharedArea.isReady = true;
12        synchronized (sharedArea) {
13            sharedArea.notifyAll(); -- 기다리고 있는 모든 스레드로 신호를 보냅니다.
14        }
15    }
16 }
```

파일을 출력하는 스레드 클래스

```
1 class PrintThread extends Thread {
2     SharedArea sharedArea;
3     public void run() {
4         if (sharedArea.isReady != true) {
5             synchronized (sharedArea) {
6                 try {
7                     sharedArea.wait(); -- 신호를 기다립니다.
8                 }
9                 catch (InterruptedException e) {
10                    System.out.println(e.getMessage());
11                }
12            }
13        }
14        System.out.println(sharedArea.result);
15    }
16 }
```

파일을 소수점 두자리까지 출력하는 스레드 클래스

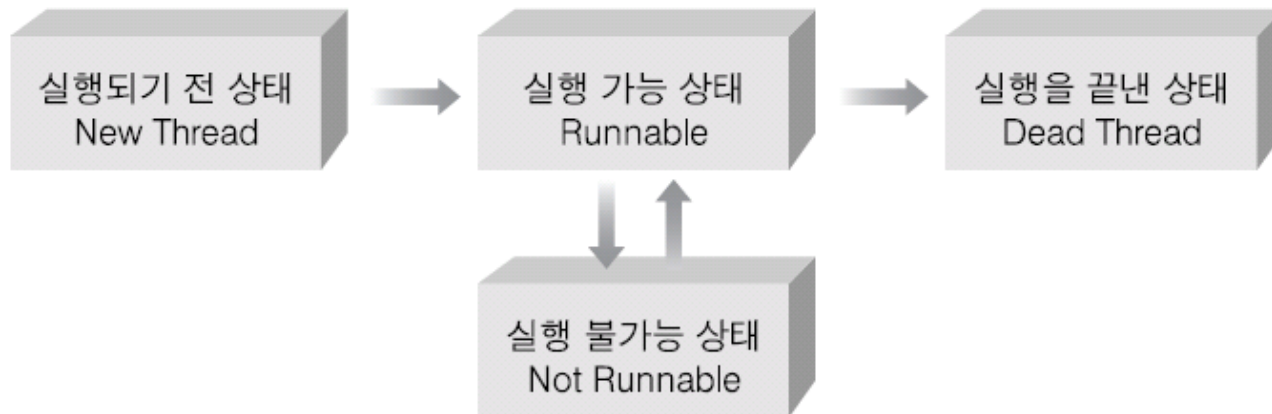
```
1 class SimplePrintThread extends Thread {
2     SharedArea sharedArea;
3     public void run() {
4         if (sharedArea.isReady != true) {
5             synchronized (sharedArea) {
6                 try {
7                     sharedArea.wait(); -- 신호를 기다립니다.
8                 }
9                 catch (InterruptedException e) {
10                    System.out.println(e.getMessage());
11                }
12            }
13        }
14        System.out.printf("%.2f %n", sharedArea.result);
15    }
16 }
```

## ● 04. 스레드의 상태

### ● 스레드의 라이프 사이클

#### ● 스레드의 라이프 사이클

- 스레드가 생성되서 start 메소드를 호출하기 전까지의 상태
- run 메소드 실행 중 상태 -> 다시 두 가지 상태로 나뉨
- - run 메소드 완료 후의 상태



## ● 04. 스레드의 상태

### ● 스레드의 상태를 알아내는 메소드

- Thread 클래스의 getState 메소드

```
Thread.State state = thread.getState();
```

열거 타입

스레드 객체

스레드의 상태를 리턴하는 메소드

## 04. 스레드의 상태

### 스레드의 상태를 알아내는 메소드

• 열거 타입 `Thread.State`의 열거값

열거 상수	의미하는 스레드의 상태
NEW	실행되기 전 상태
RUNNABLE	실행 가능 상태
WAITING	wait 메소드를 호출하고 있는 상태
TIMED_WAITING	sleep 메소드를 호출하고 있는 상태
BLOCKED	다른 스레드의 동기화 블록이나 동기화 메소드가 끝나기를 기다리고 있는 상태
TERMINATED	실행을 마친 상태

# 멀티스레드 프로그래밍

## 04. 스레드간의 상태

### 스레드의 상태를 알아내는 메소드

- [예제 18-18] 모니터링 스레드가 추가된 원주율 계

```
GV 명령 프롬프트
E:\work\chap18\18-4>java MultithreadExample9
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
쓰레드의 상태: RUNNABLE
3.141592651589258
쓰레드의 상태: TERMINATED
E:\work\chap18\18-4>
```

main 메소드를 포함하는 클래스

```
1 class MultithreadExample9 {
2     public static void main(String args[]) {
3         CalcThread thread1 = new CalcThread();
4         PrintThread thread2 = new PrintThread();
5         MonitorThread thread3 = new MonitorThread(thread1);
6         SharedArea obj = new SharedArea();
7         thread1.sharedArea = obj;
8         thread2.sharedArea = obj;
9         thread1.start();
10        thread2.start();
11        thread3.start();
12    }
13 }
```

공유 영역 클래스

```
1 class SharedArea {
2     double result;
3     boolean isReady;
4 }
```

다른 스레드를 모니터링하는 스레드 클래스

```
1 class MonitorThread extends Thread {
2     Thread thread;
3     MonitorThread(Thread thread) { // 생성자
4         this.thread = thread;
5     }
6     public void run() {
7         while (true) {
8             Thread.State state = thread.getState();
9             System.out.println("쓰레드의 상태: " + state);
10            if (state == Thread.State.TERMINATED)
11                break;
12            try {
13                Thread.sleep(2000);
14            } catch (InterruptedException e) {
15                e.printStackTrace();
16            }
17        }
18    }
19 }
```

파이를 계산하는 스레드 클래스

```
1 class CalcThread extends Thread {
2     SharedArea sharedArea;
3     public void run() {
4         double total = 0.0;
5         for (int cnt = 1; cnt < 1000000000; cnt += 2)
6             if (cnt / 2 % 2 == 0)
7                 total += 1.0 / cnt;
8             else
9                 total -= 1.0 / cnt;
10        sharedArea.result = total * 4;
11        sharedArea.isReady = true;
12        synchronized (sharedArea) {
13            sharedArea.notify();
14        }
15    }
16 }
```

파이를 출력하는 스레드 클래스

```
1 class PrintThread extends Thread {
2     SharedArea sharedArea;
3     public void run() {
4         if (sharedArea.isReady != true) {
5             synchronized (sharedArea) {
6                 try {
7                     sharedArea.wait();
8                 }
9                 catch (InterruptedException e) {
10                    System.out.println(e.getMessage());
11                }
12            }
13        }
14        System.out.println(sharedArea.result);
15    }
16 }
```



# 멀티스레드 프로그래밍

## 04. 스레드간의 상태

### 스레드의 상태를 알아내는 메소드

• [CalcThread를 모니터링하도록 수정된 예제 18-18]

```
E:\work\chap18\18-4>java MultithreadExample9
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
쓰레드의 상태: WAITING
3.141592651589258
쓰레드의 상태: TERMINATED
E:\work\chap18\18-4>_
```

main 메소드를 포함하는 클래스

```
1 class MultithreadExample9 {
2     public static void main(String args[]) {
3         CalcThread thread1 = new CalcThread();
4         PrintThread thread2 = new PrintThread();
5         MonitorThread thread3 = new MonitorThread(thread2);
6         SharedArea obj = new SharedArea();
7         thread1.sharedArea = obj;
8         thread2.sharedArea = obj;
9         thread1.start();
10        thread2.start();
11        thread3.start();
12    }
13 }
```

공유 영역 클래스

```
1 class SharedArea {
2     double result;
3     boolean isReady;
4 }
```

다른 스레드를 모니터링하는 스레드 클래스

```
1 class MonitorThread extends Thread {
2     Thread thread;
3     MonitorThread(Thread thread) { // 생성자
4         this.thread = thread;
5     }
6     public void run() {
7         while (true) {
8             Thread.State state = thread.getState();
9             System.out.println("쓰레드의 상태: " + state);
10            if (state == Thread.State.TERMINATED)
11                break;
12            try {
13                Thread.sleep(2000);
14            } catch (InterruptedException e) {
15                e.printStackTrace();
16            }
17        }
18    }
19 }
```

파이를 계산하는 스레드 클래스

```
1 class CalcThread extends Thread {
2     SharedArea sharedArea;
3     public void run() {
4         double total = 0.0;
5         for (int cnt = 1; cnt < 1000000000; cnt += 2)
6             if (cnt / 2 % 2 == 0)
7                 total += 1.0 / cnt;
8             else
9                 total -= 1.0 / cnt;
10        sharedArea.result = total * 4;
11        sharedArea.isReady = true;
12        synchronized (sharedArea) {
13            sharedArea.notify();
14        }
15    }
16 }
```

파일을 출력하는 스레드 클래스

```
1 class PrintThread extends Thread {
2     SharedArea sharedArea;
3     public void run() {
4         if (sharedArea.isReady != true) {
5             synchronized (sharedArea) {
6                 try {
7                     sharedArea.wait();
8                 }
9             }
10            catch (InterruptedException e) {
11                System.out.println(e.getMessage());
12            }
13        }
14        System.out.println(sharedArea.result);
15    }
16 }
```

영

어

하

시

다

^^!